AD_____

Award Number: DAMD17-97-C-7042

TITLE: 3D Real Time Surgical Simulator

PRINCIPAL INVESTIGATOR: Alan Sullivan, Ph.D.

CONTRACTING ORGANIZATION: Dimensional Media Associates, Incorporated
New York, New York 10011

REPORT DATE: April 2001

TYPE OF REPORT: Final

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

DISTRIBUTION STATEMENT: Distribution authorized to U.S. Government agencies only (proprietary information, Apr 01). Other requests for this document shall be referred to U.S. Army Medical Research and Materiel Command, 504 Scott Street, Fort Detrick, Maryland 21702-5012.

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision unless so designated by other documentation.

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE U.S. GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

LIMITED RIGHTS LEGEND

Award Number: DAMD17-97-C-7042
Organization: Dimensional Media Associates, Incorporated

Those portions of the technical data contained in this report marked as limited rights data shall not, without the written permission of the above contractor, be (a) released or disclosed outside the government, (b) used by the Government for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Government, except that the Government may release or disclose technical data to persons outside the Government, or permit the use of technical data by such persons, if (i) such release, disclosure, or use is necessary for emergency repair or overhaul or (ii) is a release or disclosure of technical data (other than detailed manufacturing or process data) to, or use of such data by, a foreign government that is in the interest of the Government and is required for evaluational or informational purposes, provided in either case that such release, disclosure or use is made subject to a prohibition that the person to whom the data is released or disclosed may not further use, release or disclose such data, and the contractor or subcontractor or subcontractor asserting the restriction is notified of such release, disclosure or use. This legend, together with the indications of the portions of this data which are subject to such limitations, shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

_____ _____

_____ _____

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2001	Final (25 Sep 97 - 31 Mar 01)

4. TITLE AND SUBTITLE
3D Real Time Surgical Simulator

5. FUNDING NUMBERS
DAMD17-97-C-7042

6. AUTHOR(S)
Alan Sullivan, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dimensional Media Associates, Incorporated
New York, New York 10011

E-Mail: as@3dmedia.com

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
This report contains colored photos

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution authorized to U.S. Government agencies only (proprietary information, Apr 01). Other requests for this document shall be referred to U.S. Army Medical Research and Materiel Command, 504 Scott Street, Fort Detrick, Maryland 21702-5012.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

The purpose of this program of research was the development of a high performance Multiplanar Volumetric 3D Display (MVD) to serve as the primary information display for an Advanced Volumetric Surgical Simulator (AVSS). The future AVSS will combine together a Multiplanar Volumetric Display with a pair of force feedback haptic interfaces to form a general purpose tactile task simulation platform upon which to run a wide range of surgical simulation software. The MVD is a true 3D display which produces full color, images having all of the visual attributes of real objects. This research effort was successful in nearly all of its goals which included development of new display materials, extension of the compatible computer platforms, improvement to the programming interface, and development of a custom high speed video projector. Minor shortfalls in performance of a key component of the video projector mandate that an additional prototype generation will be required before commercialization of the MVD technology can take place. This final generation has been funded through internal funds and is nearing completion as of the date of this report. Commercialization of the MVD technology will allow the development of the AVSS as well as many other military and civilian 3D display applications.

14. SUBJECT TERMS	15. NUMBER OF PAGES
3D, display, volumetric, multiplanar, three-dimensional	152
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Table of Contents

I. INTRODUCTION

The purpose of this program of research was the development of a high performance Multiplanar Volumetric 3D Display (MVD) to serve as the primary information display for an Advanced Volumetric Surgical Simulator (AVSS). The future AVSS will combine together a Multiplanar Volumetric Display with a pair of force feedback haptic interfaces to form a general purpose tactile task simulation platform upon which to run a wide range of surgical simulation software. The goal of this research was to develop, integrate and demonstrate the components and software application programming interface (API) of the MVD.

II. BODY OF REPORT

A Multiplanar Volumetric Display is a true three dimensional display system that creates images having all of the viewing characteristics of real objects. Figure 1 shows a schematic diagram of the MVD. The MVD is essentially a rear projection 3D display. The overall system consists of a computer connected through a high speed data link to an external 3D framebuffer. The framebuffer in turn feeds data at a high rate to a high speed video projector. The images from the video projector are projected into an electronic variable-depth projection volume known as the Multiplanar Optical Element (MOE). The MOE converts the high speed stream of 2D images from the video projector into a single 3D image by stopping each 2D image at the appropriate depth. Proprietary Multiplanar Anti-aliasing algorithms smooth the transitions between image slices to create continuous appearing 3D images. The resulting 3D image can be viewed directly within the MOE, just like looking at fish in a fish tank, or may be projected out into free space by a real image projection optical system. The projected image is free from the volume of the MOE and, therefore, can be touched, felt and interacted with via a co-aligned haptic interface, making development of a surgical simulator, or other tactile task trainer, possible.

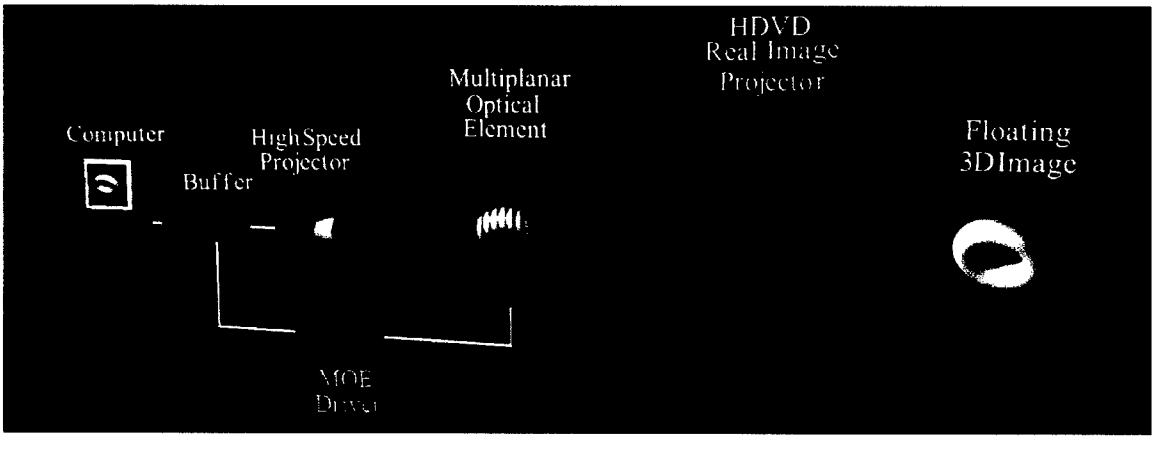


Figure 1. Schematic diagram of the Multiplanar Volumetric 3D Display showing the computer system feeding data to the high speed projector whose sequence of 2D image slices are converted to a 3D image by the Multiplanar Optical Element. The real image projector, in turn, creates a floating image.

The three dimensional image created by the Multiplanar Volumetric Display is truly 3D, not an optical illusion. The points of light that compose the image (voxels) are actually located in space where they appear. This means that 3D images created by a Multiplanar Volumetric Display have all of the characteristics of real objects. They have a wide field of view, and can be "looked around", in both the horizontal and vertical directions. They require the viewer's eyes to correctly alter focus and convergence as their gaze shifts from the front to the back of the image. The 3D image is visible to a number of viewers at any distance, each with the appropriate perspective view. Unlike other volumetric display systems, the HyperCube has no moving parts, does not generally require laser light, and is highly compatible with existing display components and manufacturing methodologies. This last point is critical to rapidly transition the technology from Research and Development to commercial manufacturing and sales.

A number of key component requirements emerge from the preceding description. The high speed video projector must be fast enough to project the 2D images slices that make up the 3D image at a rate high enough to avoid image flicker in the 3D image. And the MOE planes must be highly transparent in the transparent state, high scattering in the scattering state, and make rapid and clean transitions between these two states.

There will be references throughout this document to the refresh rate, field rate and update rate of the display. The refresh rate is the rate at which the entire volume image is produced. This rate must be greater than 35 Hz, and preferably at least 50 Hz, to avoid image flicker. The field rate as the rate at which each 2D image slice, or field, is projected into the MOE. For example, in a 12 plane MVD operating at a refresh rate of 41.7 Hz requires a field rate of 500 Hz. The update rate is the rate at which new 3D content can be transferred from the computer to the MVD. For realistic and responsive visual simulator this rate must be at least 15 Hz, but preferably greater than 30 Hz.

Table 1 lists the performance specifications of the prototype Multiplanar Volumetric Display developed under this program. Overall the project was very successful. There were nine distinct improvements in the MVD technology over the previous generation. These were:

a. Development of both SGI and PC interface capabilities

b. Development of the MVD Application Programming Interface (API) software

c. Identification and adoption of a high speed data transfer interface technology

d. Development of a custom MVD framebuffer incorporating Multiplanar Anti-aliasing in hardware

e. Customization of analog ferroelectric liquid crystal on silicon (aFLCoS) spatial light modulators for use in a high speed video projector

f. Development of a projection engine architecture to convert the spatial light modulator into a video projector

g. Development of improved MOE materials

h. Development of large MOE cells

i. Development of a MOE driver

Of all of these improvement, only the customization of the light modulator reached less than 100% of its performance goals. This component is a 512 x 512 pixel analog ferroelectric liquid crystal on silicon (aFLCoS) spatial light modulator from Boulder Nonlinear Systems, Inc. (Boulder, CO) capable of 500 DC-balanced fields per second. As will be discussed further in section C.3, the development of the custom aFLCoS SLM was the most difficult and time consuming task of the project and led to an appreciable extension of its duration. Ultimately, the performance of this component, in terms of uniformity and contrast ratio, had a significant negative impact on the visual performance of the MVD, leading to another iteration of display development before DMA can commercialize the MVD technology. The performance issues of the SLM will be discussed in greater detail in section C.3.

MVD Transverse resolution	512 x 512 pixels
Light modulator	Analog Ferroelectric Liquid Crystal on Silicon (aFLCoS)
Color depth	21-bits (2,097,152 colors)
Number of MOE planes	12
Total number of image voxels	3.14 million
Image volume	15" x 13" x 3.1"
Image refresh rate	41.7 Hz
Image update rate (plane transfer)	14 Hz

Table 1. Performance specifications of the Multiplanar Volumetric Display developed under this project.

Statement of Work from Original Proposal

The statement of work from original proposal described the following tasks:

Task 1 – MOE improvements

The MOE optical and electrical characteristics will be improved with particular attention paid to issues of clear state transparency, off-axis haze and switching speed. The research will focus on improvements in the PDLC materials for the MOE.

Task 2 – Volumetric real image projector

A carefully designed volumetric projection system will be engineered to provide optimal characteristics for the application of projecting three dimensional images from the MOE. Field of view and ergonomic consideration will be addressed.

Task 3 – High Frame Rate Image source (video projector) development

DMA will develop a high frame rate image source based upon Texas Instruments Digital Light processing (DLP) technology. The image source will be required to produce images of at least 15-bit color, with a frame rate greater than 1,000 field per second. The architecture of the DLP engine will be redesigned to permit direct integration with a high performance graphics computer digital interface.

Task 4 – High speed data link

DMA will develop high speed three-dimensional image processing hardware and software based upon Silicon Graphics hardware to deliver data to the high frame rate image source. The task will involve transforming a three-dimensional OpenGL data set into a multiplanar data set and delivering the information to the image source in real-time. It also involves the design and construction of a high bandwidth interface between the computer system and the image source.

Task 5 – MVD API software and display integration

Integration will consist of physically connecting the computer system and the high frame rate image source, establishing proper communications, and integrating software elements.

It is conceptually more understandable to discuss the results of this work in the order of the data flow from computer to the viewer, rather than in the order given above.

A. TASK 5a – MVD API software

The MVD can be driven by either a personal computer (PC) running the Window NT 4.0 operating system or by a Silicon Graphics workstation. The PC currently used is a Dell computer with dual 833 MHz Pentium III processors and equipped with a Nvidia GeForce2 3D graphics card that is controlled through the standard OpenGL graphics API. The SGI workstation is an Octane SSI with dual 195 MHz R10000 RISC processors and a PCI card interface.

The MVD application programming interface (API) is a library of routines used to add MVD functionality to computer source code. The MVD API was written for the PC in Pascal using Borland's Delphi IDE (see Appendix A) and in C++ for the SGI. It consists of the following commands:

a. *mvdSet_state* – sets the operating parameters of the MVD including the number of planes, transverse resolution of each plane, and a range of flags to control various MVD operating modes.

b. *mvdInit* - initializes both internal memory space and the high speed data transfer card.

c. *mvdReadgl* - reads the color and z-buffer data from the graphics processor's framebuffer back into main memory.

d. *mvdFormat_data* - combines the separate color and z-buffer data into a 32-bit word per pixel (see Appendix B for details of the 32-bit data word format).

e. *mvdTransfer* - performs handshaking with the framebuffer electronics to confirm readiness to receive data and, if confirmed, initiates the data transfer by the data transfer card and then monitors the process for completion.

f. *mvdClose* – frees internal memory space and shuts down the data transfer card.

The API functions, mvdSet_state and mvdInit, are generally called during an application's initialization phase. mvdSet_state is an overloaded function that takes as input a parameter flag and a parameter value. It is called at least three times to initialize the MVD state variables defining the number of MOE planes, and the transverse display resolution. Following the calls to mvdSet_state, mvdInit is called to initialize the MVD hardware. mvdInit returns an integer to indicate either success or the reason for initialization failure. Future API releases will write the MVD hardware parameters (number of planes, and transverse resolution) to the MS Windows registry during installation.

3D images for the MVD are created in the computer in the standard way. The programmer defines 3D polygonal surfaces, texture maps, color and lighting models through conventional OpenGL calls. The result is a 2D image on the computer monitor that is created by the graphics hardware through its rendering processor. Associated with the image on the monitor is both the color data and an associated depth map (z-buffer) in the framebuffer of the graphics hardware. The z-buffer has the same x, y resolution as the color image and is the depth, in scaled coordinates, of each pixel in the color image.

The API functions, mvdReadgl, mvdFormat_data, and mvdTransfer are called following the end of the conventional rendering process. In OpenGL this occurs after the glSwapBuffers command is issued. These MVD API commands are called in sequence to create an image in the MVD. Following the return from mvdTransfer, the entire rendering process can begin again. Given the explicit nature of writing images to the MVD it can be updated as frequently or infrequently as the programmer may desire.

It should be noted that the choice of the 3D graphics card can strongly affect the display update rate by affecting how long it takes to transfer data from the framebuffer back to main memory. Although any OpenGL graphics card can be used not all graphics cards will provide good update rates. The Nvidia GeForce2 graphics card currently in use can transfer both the color and depth framebuffer data from a 512 x 512 window to main memory in 11.7 msec, more than fast enough for realtime simulation. The framebuffer read back time can vary widely depending on the choice of graphics card.

Two different data transfer modes from computer to MVD are currently supported by both hardware and API. These are plane mode and block mode, and are established through the mvdSet_state function. In plane transfer mode the number of data words

transferred to the MVD is equal to the number of pixels in a single 2D image slice, 512 x 512 in the current hardware. The pixels are sent in order from the top, left to the bottom, right with each data word giving the RGB color and depth of each pixel, with the depth scaled by the API to be a fixed point value in the range from 0 to N_{Planes}. The depth data is used by the input processor of the framebuffer to determine the MVD plane, or planes, on which the data belongs. In this way only one voxel at any depth at a particular x, y location is illuminated. The result is 3D image that is either a single 3D surface, a 3D wireframe, or a 3D point cloud. Multiple 3D surfaced images cannot be produced in plane transfer mode.

In block transfer mode, the data for every voxel in the display is transferred from the computer to the display, including voxels whose color is black. In the current prototype, this constitutes the transfer of 512 x 512 x 12 data words, obviously taking 12 times longer to transfer to the display than a plane transfer mode image. In block mode, the depth information in the data words is ignored since the ordering of data words determines their x, y and z location within the display. Block mode allows multi-surface 3D images and space filling volumetric data, such as MRI or CT data, to be displayed in the MVD.

B. TASK 4 - High Speed Data Link to Image Source

A high speed data transfer card from National Instruments, Inc. (Austin, TX) is used to transfer 3D images to the MVD. This card (PCI-DIO-32HS) is capable of directly reading data out of main memory through a process known as direct memory access (DMA), and can transfer 32-bit parallel data words from the computer to a framebuffer at up to 76 MBytes per second. Accordingly, the card can transfer a plane transfer mode image from the computer to the framebuffer in 13.7 milliseconds, and a block mode formatted 512x512x12 voxel image in 164 milliseconds. Appendix C gives the pin assignments for the connection between the data transfer card and the MVD framebuffer.

The data transfer card is available with device drivers for both Apple Macintosh computers and Wintel PCs. An objective of this project was to enable the MVD to also operate with Silicon Graphics workstations. Therefore, a device drive was written by DMA to provide this additional connectivity. The source code of the device driver is attached as Appendix D.

C. TASK 3 - High Frame Rate Image Source

The high frame rate image source consists of an external multiplanar framebuffer and the high speed video projector. The video projector consists of the spatial light modulator and the optical projection engine, which is further sub-divided into the illumination subsystem, color separation optics, color combination optics and projection lens..

C.1. MVD framebuffer electronics

The MVD framebuffer was custom developed by DMA for this project. It is shown conceptually in figure 2 and complete framebuffer schematics are included in Appendix

E. It consists of three identical printed circuit boards, one for each color channel of the high speed video projector. The cable from the data transfer card terminates on the red framebuffer board which has been designated as the master controller. The green and blue brightness data and the depth data is transferred from the red board to the green and blue slave boards through two short cables identical to the input cable. Data clock, input, and output synchronization signals are sent from the red board to the green and blue boards through BNC cables.

Each board consists of an input processor, multiplanar framebuffer and output drivers. The input processor is a field programmable gate array (FPGA) that receives the pixel data from the data transfer card, performs multiplanar antialiasing on the pixel data (more on this later) and then places the processed data into the multiplanar framebuffer. The multiplanar framebuffer consists of 25 Mbytes of dual ported memory arranged in a double buffer architecture. The double buffered architecture allows one 12.5 MByte sub-buffer to written while the other sub-buffer is being display, and permits a high display refresh rate in the presence of a lower and variable 3D image update rate. Essentially, the framebuffer receives data from the computer serially at 76 Mbytes/sec and outputs it in parallel to the video projector at a rate of 1000 Mbytes/sec/color. The framebuffer is large enough to hold 50 two dimensional images, each 512 x 512 pixels and 21-bit color.

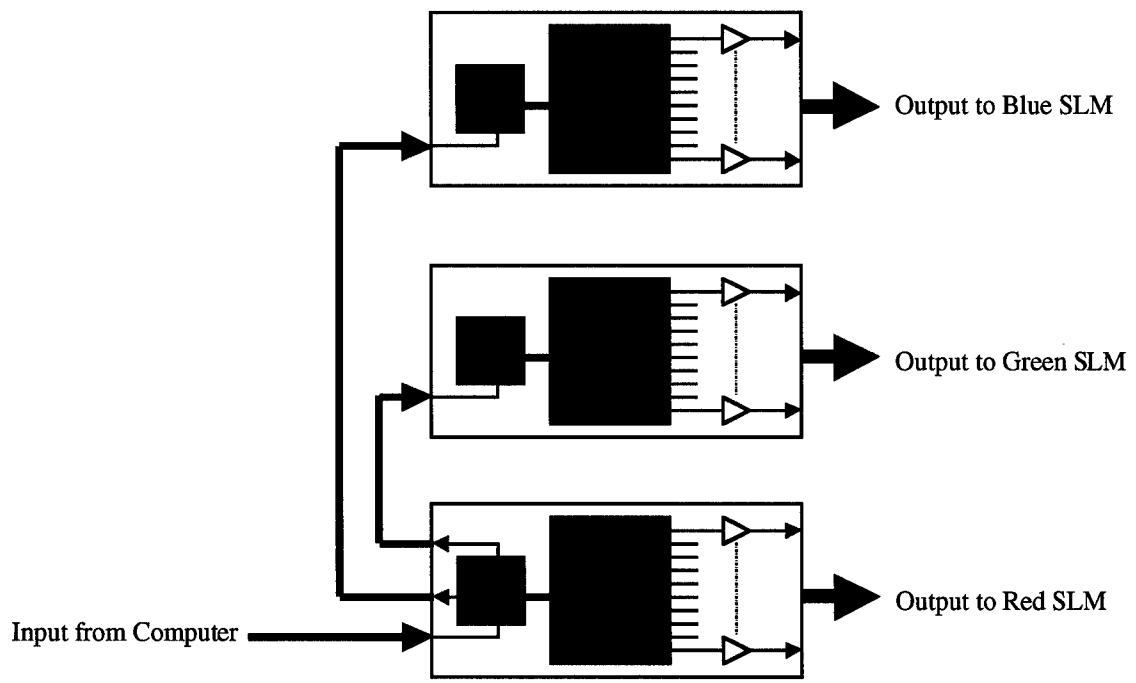


Figure 2. Schematic diagram of MVD framebuffer consisting of three identical boards, one for each color. The red board acts as the master and transfers brightness, depth and synchronization signals to the green and blue boards. IP is the input processor, FB is the framebuffer.

The output driver section of each board consists of 32 digital-to-analog converters (DAC) arranged in parallel, each delivering data to a 16 column x 512 row sub array of the light modulator of the high speed video projector. The DACs are equipped with internal RAM that is used as a look-up-table (LUT) used to linearize the nonlinear

voltage to brightness transfer function of the aFLCoS spatial light modulators. The DAC receives an 7-bit brightness value from the MVD framebuffer, uses it as an address to retrieve a 10-bit value from the LUT, then converts the 10-bit value to a current in the range of 0 milliamps to 13.62 milliamps. The current signal from the DACs is delivered to each SLM through a 100 conductor ribbon cable that terminates in a small printed circuit board. The board has 8 quad op-amps that are configured as current to voltage converters. The resulting voltage range is 0 volts to 4.32 volts. These voltages are applied to the input pins of the SLM. Current signals, rather than voltage, are used to provide a high level of noise immunity. The data clock to the DACs is 33 MHz.

Although the 3D image within the MOE actually consists of a stack of discrete 2D image slices, the image can be made to appear continuous through the use of a processing algorithm developed previously be DMA called multiplanar anti-aliasing. Multiplanar ant-aliasing eliminates the abrupt transitions between MVD image slices by effectively creating voxels that appear to the viewer to be between two MOE planes.

To use multiplanar anti-aliasing the voxel depth data is transferred to the framebuffer as a fixed point number consisting of a 6-bit integer portion and a 5-bit fractional portion. The integer portion determines the pair of planes on which the brightness data is written and the fractional portion determines their relative brightness'. For example, if the integer portion is 5 the brightness data will be written to planes 5 and 6. If the fractional part is 0 then 100% of the brightness is on plane 5 and 0% of the brightness is on plane 6. If the fraction part is 16 then 50% of the brightness is on plane 5 and 50% on plane 6. For the maximum fractional value of 31, plane 5 gets 1/32, or 3.1% and plane 6 gets 31/32, or 96.9%.

The antialiasing calculation is carried out on-the-fly by the input processor. For each data value transferred the brightness and the fractional value of the depth is used to compute two new brightness values. These values are written to the portions of the framebuffer memory determined by integer portion of the depth value.

C.2. Projection Engine

The original design for the projection engine of the high speed video projector is shown schematically in figure 3 and the final design is shown in figure 4. The original design employed a 500 Watt Cermax Xenon arc lamp (Model # EX500-13F) from ILC Technology (Sunnyvale, CA, now a division of EG&G) as the light source and is similar to the design described in reference 1. The light from the arc lamp is focused on an 8 mm x 8 mm x 100 mm BK7 integrating rod to produce uniform illumination. The light from the output face of the integrating rod is relayed to the light modulators by relay lenses 1 and 2 and the field lenses. Along the way the light is split by dichroic mirrors from OCLI (Santa Rosa, CA) into red, green and blue channels. These are routed by turning mirrors toward the appropriate light modulator. After passing through the field lens the light is incident on a wide bandwidth, wide angle polarizing beam splitter (PBS) from OCLI. The vertically polarized component of the light is reflected 90° by the PBS, passes through a zero-order half wave plate from Newport Corp. (Irvine, CA), and illuminates the light modulator with an image of the end of the integrator rod. The half wave plate allows the

polarization axis of the light to be rotated for optimization of the light modulator contrast ratio. The light modulator reflects the light back toward the PBS and rotates the polarization depending on the voltage supplied to each pixel. Rotation by 90° cause the light to be transmitted by the PBS toward an x-cube beam combiner from Balzers Thin Films (Golden, CO). The x-cube combines red, green and blue light entering through three faces into a single full color beam exiting the fourth face. This light is imaged by a 55 mm F.L., 100 mm B.F.L., F/2.5 projection lens from Buhl Systems. The large back focal length provides the space needed for the x-cube and the PBS between the SLM and the projection lens. The projection lens can be focused between 0.5 meters and 5 meters thereby accommodating a wide range of MOE sizes.

Each light modulator is illuminated by a uniform 8 mm x 8 mm light beam having an FWHM angular range of ± 15° (F/1.9) and a bandwidth of approximately 90 nm centered at 455 nm for the blue, 545 nm for the green and 635 nm for the red.

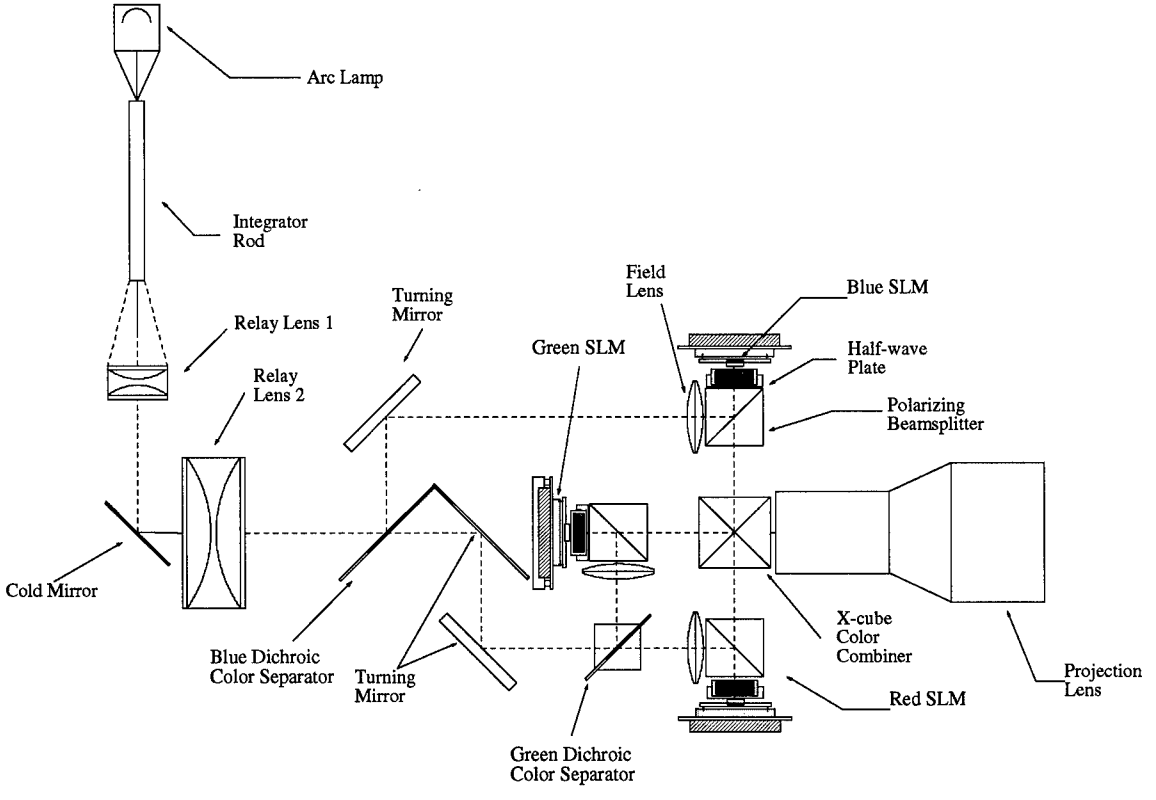


Figure 3. Schematic diagram of the original design of the high speed video projector. The heart of the illumination system is a 500 watt Cermax xenon arc lamp.

Limitations in the performance of the spatial light modulator when illuminated by broad band, low F/# light forced the redesign of the projection engine to that shown in figure 4. These limitations, described in section C.4 caused the 3D images produced by the original design to have low contrast in the range of 30:1. By comparison, the contrast of commercial video projectors is in the range of 400:1.

The design of figure 4 has the same PBS, half wave plates, x-cube color combiner and projection lens as the original, but has the arc lamp based illumination sub-system is replaced by one that uses collimated, monochromatic laser light to illuminate the light modulator.

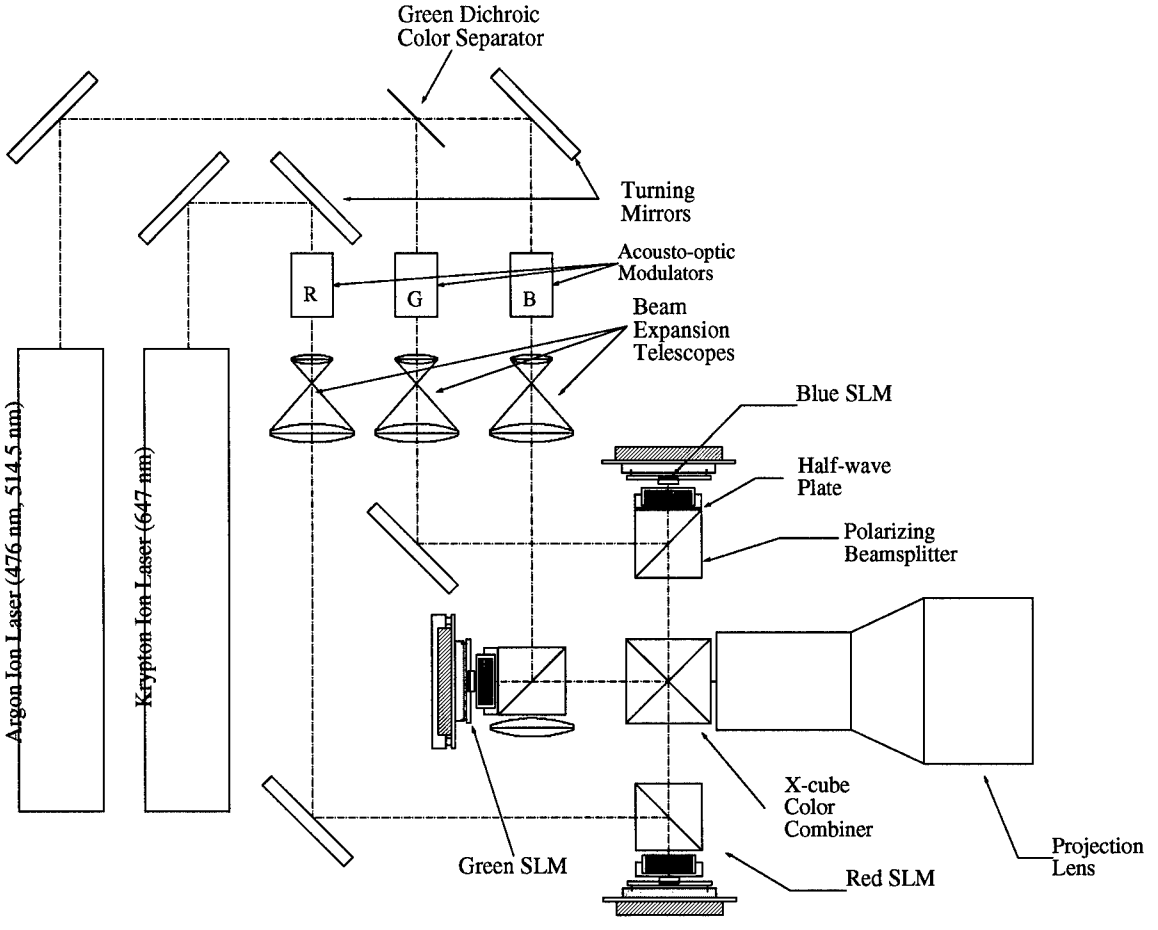


Figure 4. Schematic diagram of the final design of the high speed video projector. The original Cermax xenon arc lamp has been replaced by lasers to provide illumination.

Full color illumination is provided by a combination of an argon ion laser, simultaneously emitting green light (514.5 nm) and blue light (476 nm), and a krypton ion laser that emits red light (647 nm). The system also employs an acousto-optic light modulator for each color to blank the laser light during the 60% of the field period associated with DC-balancing of the light modulator. The result was a dramatic improvement to the projector contrast ration to over 250:1 for the green and red channels and over 150:1 for the blue channel, adequate to demonstrate the feasibility of the rest of the display design, but inadequate for commercialization. Again the limitation in performance is caused by the spatial light modulators.

C.3. Spatial Light Modulator

The original statement of work indicated that DMA would attempt to use the Texas Instruments' DLP light modulator technology in the prototype MVD. Unfortunately, at the time TI did not allow third parties to develop modified versions of its DLP technology for novel applications, thereby preventing its use in the MVD. Fortunately, a reasonable replacement technology was identified from Boulder Nonlinear Systems, Inc. (BNS, Boulder, CO).

The three spatial light modulators in the high speed video projector are custom components developed by BNS for this project. They are reflective liquid crystal on silicon (LCoS) light modulators in which a silicon wafer is fabricated through CMOS processing into an 2D array of reflective aluminum pixel electrodes with associated storage capacitors and electronic switches to form a silicon backplane (ref. 2, 3). Onto this backplane is applied a liquid crystal alignment layer (often rubbed polyimide), a 2-5 micron thick layer of liquid crystals, and a cover glass with a transparent conductive layer of indium tin oxide (ITO) to serve as a counter electrode. The maturity and resolution of CMOS processing allows a large number of backplanes to be produced simultaneously on a silicon wafer, each with a large number of pixels (up to 2400 x 2048) with small sizes (down to 5 microns) and with a high fill factor (up to 95%) to be fabricated at low cost and with high yield.

Although conventional LCoS light modulators, such as those from Three Five Systems, Inc., may use the type of nematic liquid crystal found in LCDs, these are far too slow (~ 10 msec) for the high field rate requirements of the MVD (~100 microseconds). BNS produces LCoS light modulators that use ferroelectric liquid crystals aligned in such a way as to produce analog grayscale optical modulation as a function of pixel electrode voltage.

There are a number of important requirements for LCoS light modulators that are used in video projectors (ref. 4). They must have a high fill factor (the fraction of a pixel's area that reflects light) in order to provide a high optical efficiency. The aluminum used for the pixel electrodes must be optically flat in addition to being a good conductor so that there will be a good specular reflection. The entire CMOS die of the light modulator must be globally flat (planarized) to within a fraction of a wavelength of light to provide a uniform thickness of the liquid crystal layer. And the CMOS circuitry under the pixel electrode array must be shielded from illumination by the incident light which would otherwise cause the pixel voltages to be discharged.

At the beginning of this project BNS had an existing analog ferroelectric spatial light modulator that was used for optical computing applications. Because they are not intended for projection applications, these SLMs have a low fill factor, rough aluminum pixel electrodes, and poor die planarity. BNS was involved in a project to apply a sacrificial layer to the top of these die, planarize the sacrificial layer, and then apply a spatially registered array of high reflectance, high fill factor pixel electrodes. DMA was going to procure three of these SLMs to use in the MVD high speed video projector.

Unfortunately, the effort was not successful as the resulting SLMs were not only not highly reflective and well planarized, but did not function electrically.

DMA then contracted with BNS to develop a custom analog ferroelectric LCoS SLM having the projector component characteristics described above as the starting principles of the CMOS design. The resulting design was then fabricated at a CMOS silicon foundry into five 8" diameter wafers containing a total of roughly 1500 usable die. Each die has 512 x 512 pixels on 15 micron centers with an 85% fill factor with the pixel electrode area occupying 7.68 mm x 7.68 mm. After dicing a wafer into individual die, the final SLM is completed as described above. Figure 5 shows microphotographs of the one of the SLM die as it appears on the 8" silicon wafer.

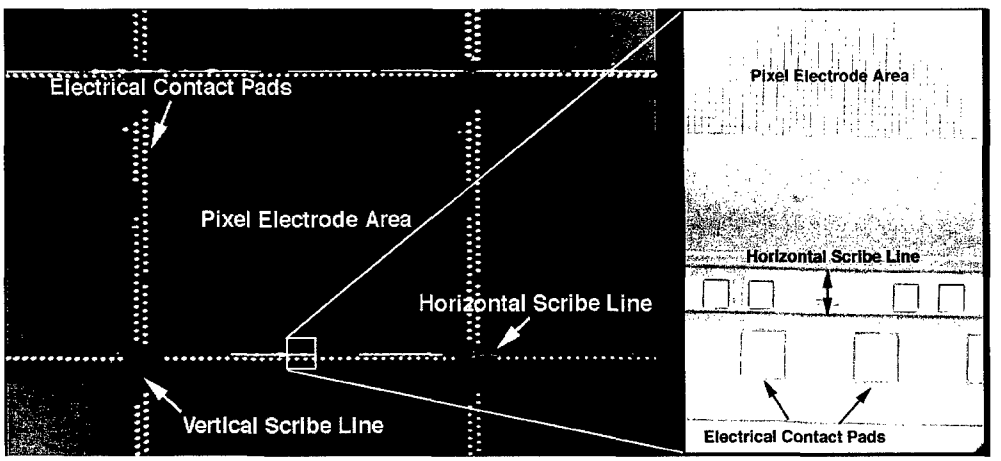


Figure 5. Microphotographs of the 8" wafer of CMOS silicon backplanes of the BNS custom analog ferroelectric light modulator. The image on the left shows several adjacent SLM die while the image on the right is a zoom in of the indicated area showing the individual 15 micron pixel electrodes. The color in the right hand image is an artifact of the microscope.

A number of custom aFLCoS light modulators were evaluated before identifying three components that provided good performance at the red, green and blue laser wavelengths used. Graphs of their optical signal verses SLM pixel voltage are shown in figure 6. In each case the half wave plate angle is adjusted to provide the best overall black level at some pixel voltage. The data is then used to compute LUT that is loaded into the framebuffer DACs in order to linearize the optical signal as a function of the digital grayscale value from the computer.

The final image of the high video projector and hence the MVD was found to suffer from the relatively low contrast ratio and low uniformity of the BNS SLMs. These problems are caused by an insufficient uniformity of the alignment of the liquid crystals that is, in turn, is caused by insufficient planarization of the silicon backplane. This type of problem is common in LCoS light modulators (ref. 4) and is caused by the unavoidable non-uniformity of the density of the semiconductor structures created during CMOS photolithography. The pixel array area of the SLM has a high density of structures whereas the shift registers at the periphery of the pixel array have a low density. The result is that the planarization process erodes the edges of the light modulator more that the center. This creates difficulty in producing a high quality of liquid crystal alignment

as well as variations in the thickness of the liquid crystal layer. The affect of these issues on the final 3D image was most severe when using incoherent, wide bandwidth, wide angle arc lamp illumination. Use of narrow band, collimated laser illumination dramatically improved the light modulator contrast ratio allowing the prototype display to function at a reasonable, but still sub-commercial, level.

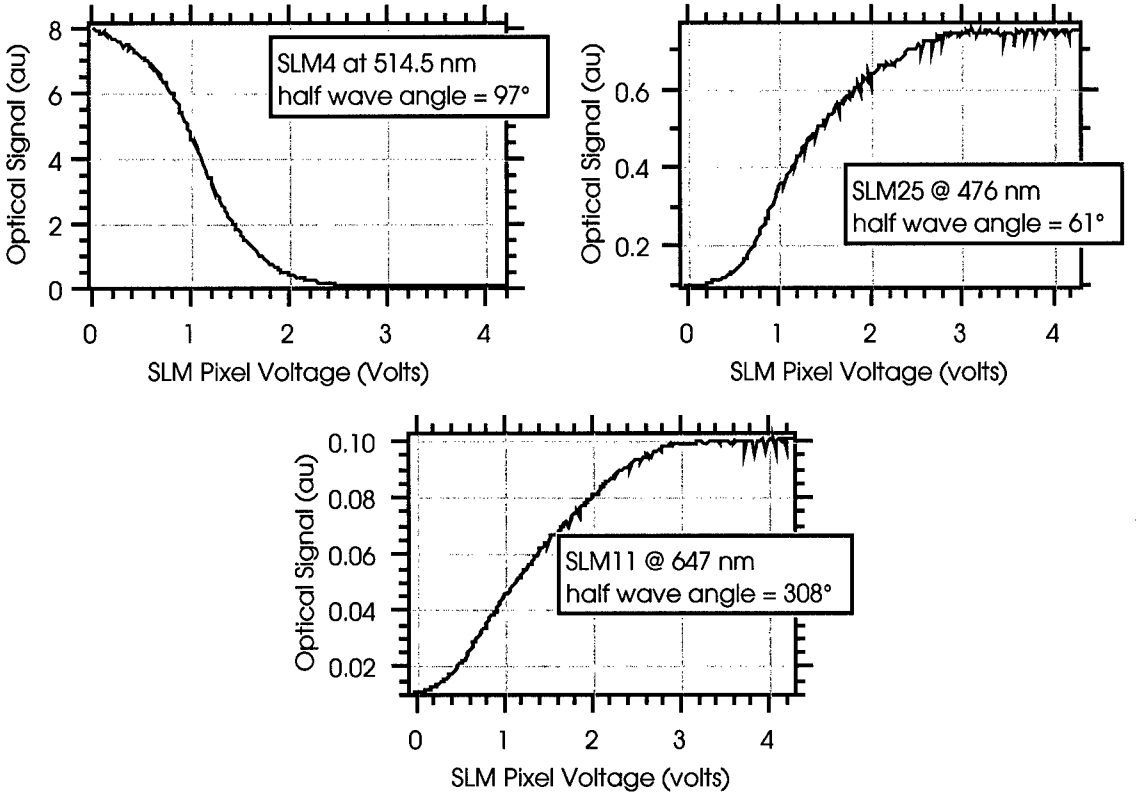


Figure 6. Graphs of the optical signal verses SLM pixel voltage for the red, green and blue SLMs. The graphs clearly show the grayscale behavior of the SLMs. The vertical scale is arbitrary.

The process for eliminating the light modulator defects is straightforward but expensive. It involves iterative modifications to the CMOS design followed by fabrication and testing of the resultant design. These modification entail the arrangement of dummy structures in the SLM periphery to obtain uniform planarization. Additional CMOS processing steps are also necessary to eliminate any height discontinuities between the pixel electrodes and inter-electrode spaces that can also affect alignment uniformity. Each iteration of this process can cost between $50,000 and $100,000 and take 3 months. Given this great cost, DMA has determined that near-term future MVD systems will be built using light modulators whose uniformity is well developed. Top among these is the DLP technology from Texas Instruments which will provide the light modulation technology for the next generation MVD.

D. TASK 1 - MOE Improvements

The MOE for this project was greatly improved in terms of its size and material speed over the previous generation. The previous MOE used shutter cells from

Crystaloid, Inc. (Kent, OH) that are 6" x 6" whereas the new MOE uses shutter cells that are 15.1" x 13.5" from LXD, Inc. (Kent, OH). To utilize the new cells a considerable refinement of the liquid crystal vacuum filling equipment and technique was required. A much large vacuum chamber was designed and built to accommodate the larger cells. Additionally, a vacuum manipulation system designed to fill five cells simultaneously was designed and built. Mechanical drawings of the chamber and cell manipulator is shown in Appendix F. A hydraulic cell press for squeezing the filled cells down to a uniform thickness was constructed by adding large ground steel press platens to a standard 10 ton hydraulic press (Dake Dura-Press Model Force 10). The platens are larger than the MOE cells and allow the press to uniformly compress the cells to their design thickness (14 microns). Mechanical drawings of the cell press are included in Appendix G.

The filling technique refinements included careful control of the minimum vacuum pressure, careful control of pressure ramping profile, and electrical preconditioning of cells to eliminate shorts. Special LC material handling techniques were also developed to guarantee mixture performance after being filled into cells.

Also developed for this project was a custom MOE driver. The schematics are included in Appendix H. The driver was designed to be as general purpose as possible given the early uncertainties about the voltage, timing, and MOE cell number. Accordingly the driver is designed to drive up to 50 MOE cells, at voltages ranging from 20 volts to 300 volts and with pulse widths from 0.1 milliseconds up to 9.99 milliseconds. The MOE driver is designed to drive the MOE cells with a bipolar electrical waveform like that shown in figure 7. A cell is powered into the clear state by applying a field greater than 10 volts/micron. To become scattering the voltage is rapidly brought to 0 volts. The voltage is held at 0 volts for a period of roughly 2 milliseconds (zero period duration) before being brought back to the clear state by applying a voltage of the opposite polarity as before. The use of a bipolar waveform causes the applied field to average out to zero thereby avoiding potential shutter degradation known to be associated with DC applied fields.

Figure 8 shows the transmission verses time for a typical MOE cell at a number of different zero period durations. The cell exhibits an ~1 msec period after the field is removed during which it remains highly transparent. Then the cell switches quickly and cleanly to a low transmission state in approximately 500 microseconds. The transmission remains low until the field is reapplied, after which it makes an immediate jump back to high transmission. The off-to-on switching time depends on the width of the zero period but is approximately 50 microseconds in the range of zero periods appropriate to MVD operation (FWHM ~2 msec).

To develop the LC materials for this project we subcontracted with Dr. Deng-Ke Yang of the Liquid Crystal Institute (LCI) of Kent State University. Dr. Yang performed the original published research (ref. 5) that led to the material used in the previous MVD generation and so was deemed to be the best person to help develop new materials for the MVD.

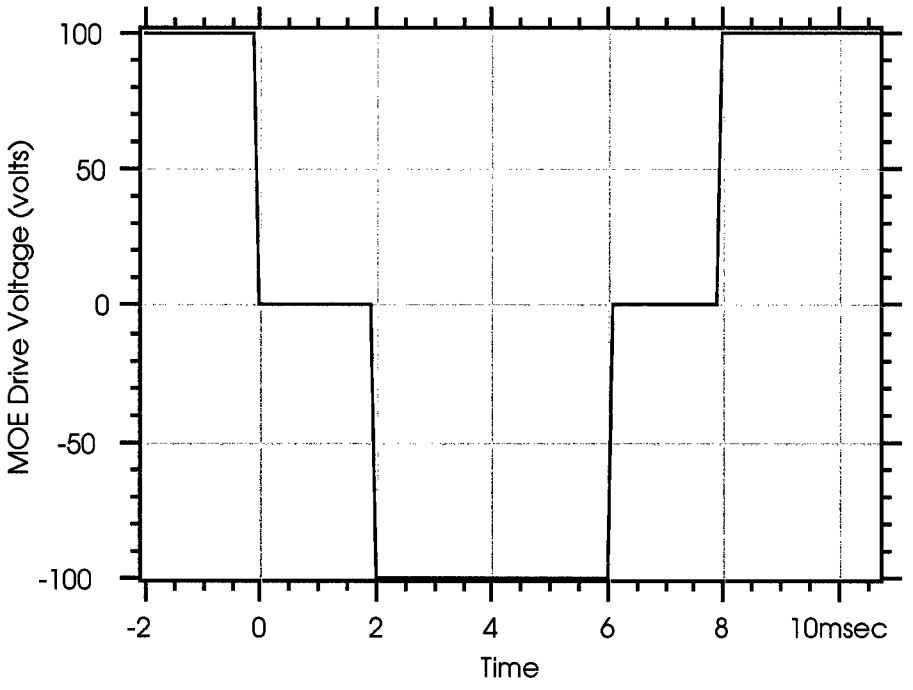


Figure 7. Bipolar drive waveform applied to MOE cells by the MOE driver. A bipolar waveform has an average field of 0 volts to avoid cell degradation caused by DC fields.

These liquid crystal materials are of a class known as polymer stabilized cholesteric textures (PSCT) in which a small percentage (~2.5% by weight) of a liquid crystal polymer is added in its unpolymerized, monomer state to a chiral nematic liquid crystal. The pitch length of the chiral nematic is chosen to be in the infrared making its interaction with visible light weak. By filling the liquid crystal/monomer mixture into the cell, aligning the liquid crystal with an applied electric field and then polymerizing the monomer, the cell become highly scattering when the field is remove. Dr. Yang spent a number of months trying a range of different nematic liquid crystals, chiral additives, and liquid crystal polymers before identifying a mixture that provides the desired electro-optic characteristics. This mixture is by no means optimized but is adequate for this generation of MVD. Future MOE work will focus on material optimization as well as engineering the multi-layer coatings within the MOE cell to minimize losses due to reflections.

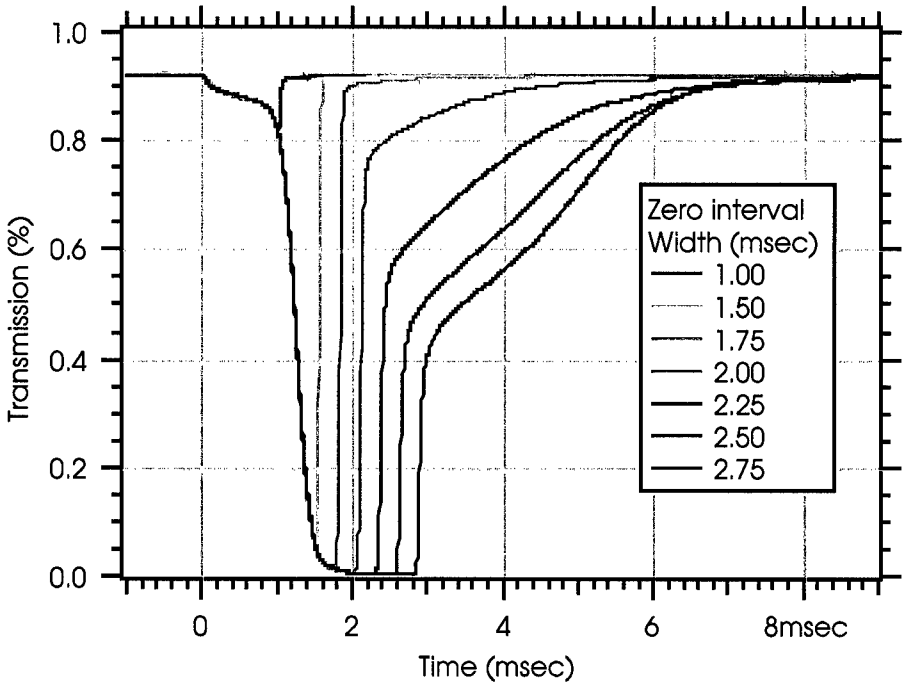


Figure 8. Transmission verses time at a number of different zero period durations for a typical MOE cell. Transmission losses are dominated by approximately 8% reflection between air-glass interfaces.

E. TASK 5b – MVD Integration

The integration of the Multiplanar Volumetric Display developed under this program took place over approximately six months. The greatest challenges of the integration process was that of carefully characterizing the pixel voltage to optical output transfer functions shown in figure 6. Each SLM must be tested separately using a linear look-up-table in the DAC section of the framebuffer. The resulting data is then inverted, scaled and loaded into the LUT to linearize the output of the SLM. Additional integration steps included careful color convergence and color balancing to achieve a reasonable white point.

Figure 9. shows a photograph of the entire integrated MVD system. Indicated in the photo are the computer, framebuffer, lasers, high speed video projector, MOE driver and both a small and large MOE. The small MOE, developed under a previous non-government program uses 12 planes, each 6" x 6" and is used for full color images since the available laser power is insufficient for full color images in the large MOE. Figure 10 shows a close-up photograph of the completed high speed video projector. Indicated are the lasers, framebuffer, light modulators, x-cube color combiner and projection lens.

Figures 11, 12, and 13 show photographs of 3D images in the small MOE. Figure 11 is a solid 3D skull while figure 12 is the same image in wireframe. Figure 13 is a brightly color 3D knot drawn by DMA to showcase the full color, grayscale capability of the display. The graininess in these images is caused by the coherent nature of the laser light. Use of arc lamp illumination in future systems will eliminate this effect.

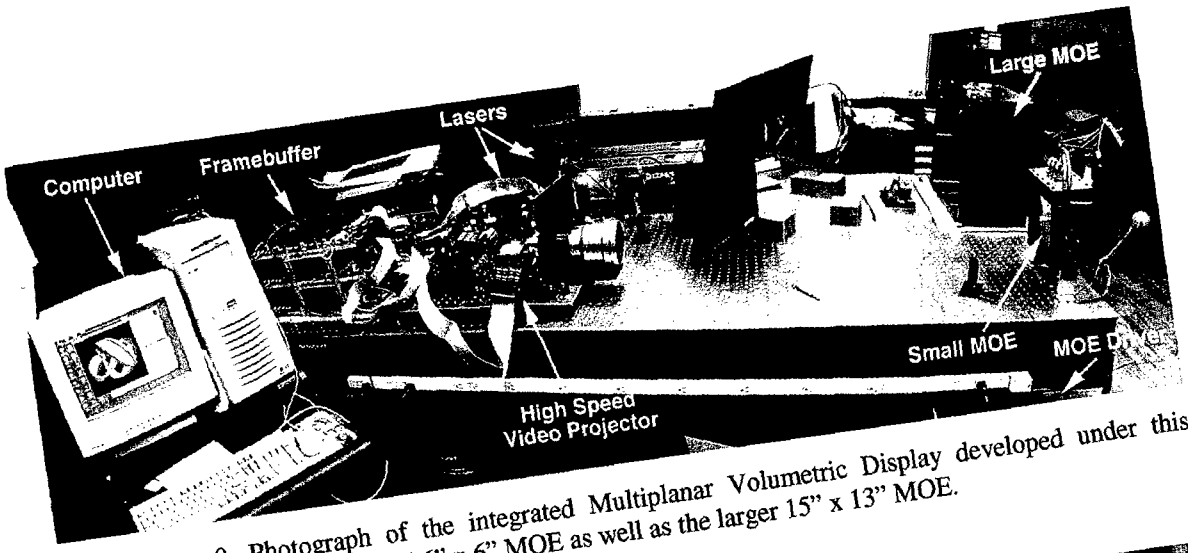


Figure 9. Photograph of the integrated Multiplanar Volumetric Display developed under this program. On the right are a small 6" x 6" MOE as well as the larger 15" x 13" MOE.

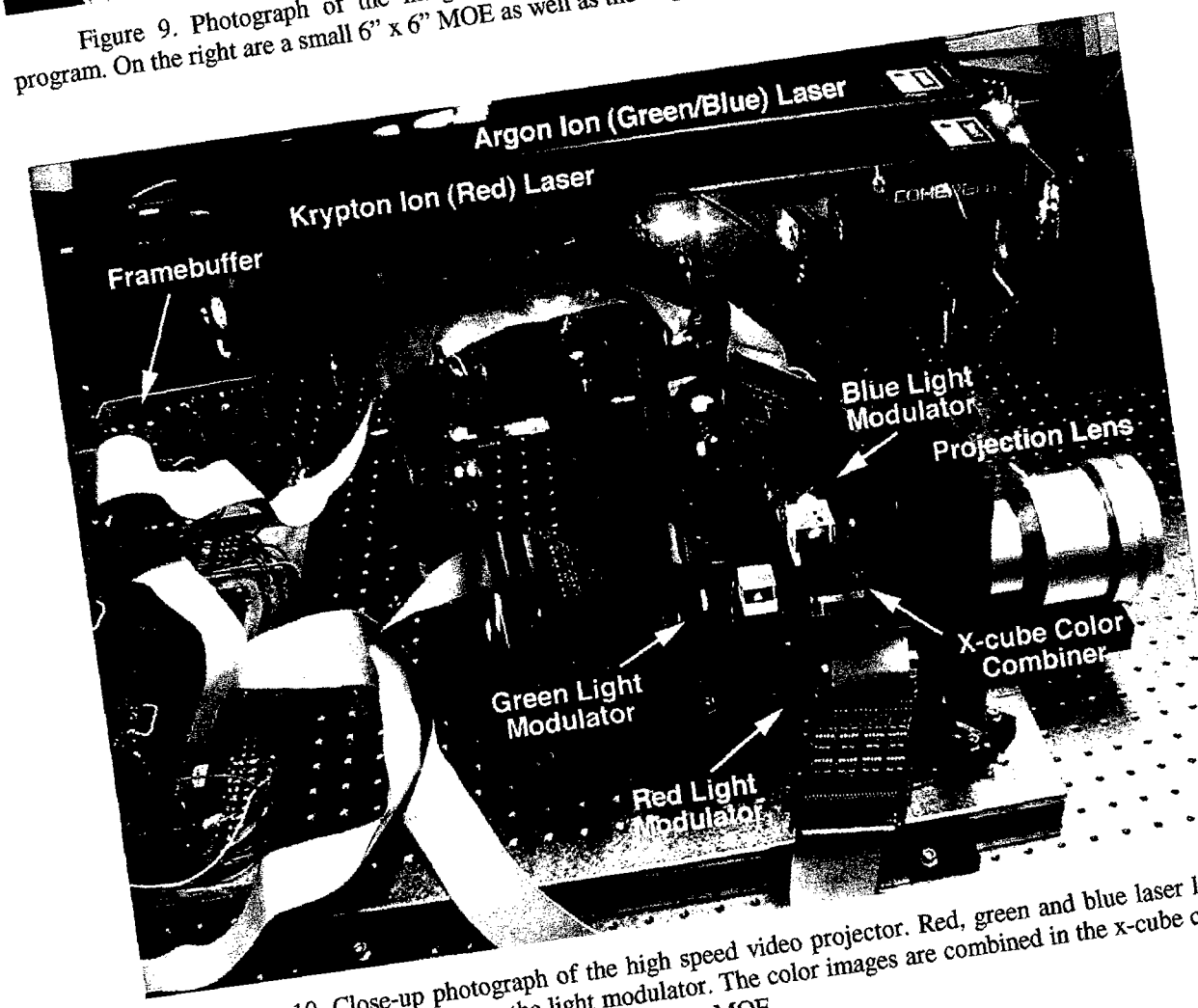


Figure 10. Close-up photograph of the high speed video projector. Red, green and blue laser light enters from above and illuminates the light modulator. The color images are combined in the x-cube color combiner and projected by the projection lens into the MOE.

Figure 11. Close-up photograph of the operating MVD. In the background to the left is the computer screen showing the conventional 2D image, while to the right is the full color 3D image in the small MOE. Limited power from the lasers prevents production of a full color image in the large MOE.

Figure 12. Close-up photograph of the operating MVD. The image is a 3D wireframe version of the same image as in figure 11.

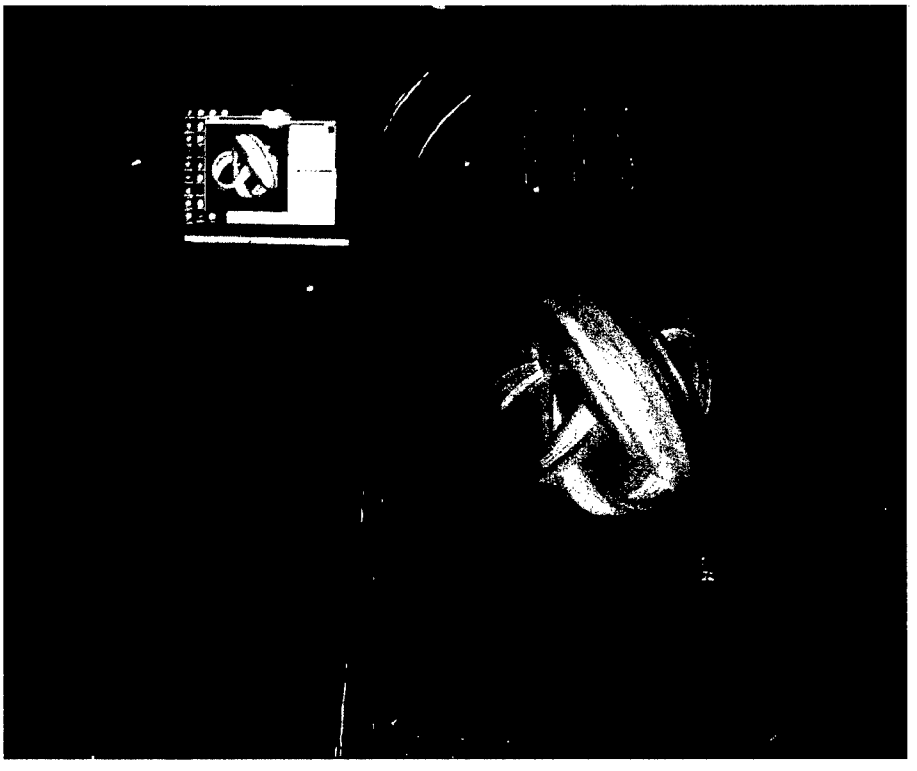


Figure 13. Close-up photograph of the operating MVD. The image is a full-color geometric knot developed to exhibit the full color performance of the display.

F. TASK 2 - Volumetric Floating Image Projector

The extensive and unexpected extra effort and expense necessary to develop the custom aFLCoS light modulator resulted in the decision not to pursue the development of real image projection optics for the floating images projector. Given the straight forward nature of this element of the project there is little risk or challenge in developing the optics at a later date.

III. KEY RESEARCH ACCOMPLISHMENTS

- Development of both SGI and PC interfaces to the Multiplanar Volumetric Display
- Development of the MVD software Application Programming Interface (API)
- Identification and adoption of a high speed data transfer interface technology
- Development of a custom MVD framebuffer incorporating Multiplanar Anti-aliasing
- Customization of analog ferroelectric liquid crystal on silicon (aFLCoS) spatial light modulators for use in a high speed video projector
- Development of a projection engine architecture to convert the spatial light modulator into a video projector

- Development of improved MOE materials
- Development of large MOE cells and the tools and methods to fill them correctly
- Development of a MOE driver

IV. REPORTABLE OUTCOMES

The most significant outcome of the project was the dramatic advancement of the MVD technology toward a commercial performance level. Although the prototype MVD described herein is not adequate for commercialization, it has played a crucial role in helping to identify the final improvement necessary to achieve it. In fact, the majority of the subsystems of the display performed in ways that can be used commercially. Only the light modulator needs to be refined. This refinement is taking place as of this writing. Texas Instruments has greatly reduced the level of restrictions on novel uses of their DLP technology allowing DMA to develop a version of the MVD using DLP. Given to excellent performance of every other subsystem of the display, it is expected that the next generation prototype will exhibit all of the performance characteristics necessary for commercialization. Following this step the Advanced Volumetric Surgical Simulator can be developed.

V. CONCLUSIONS

The result of this project was a significant advancement of the Multiplanar Volumetric Display technology toward a commercial performance level. Nearly all of the improvements in the technology undertaken during this project were successful. The one area still in need of improvement is the spatial light modulator of the high speed video projector. The solution to that is issue is being pursued currently with the expectation that by the third quarter of 2001 an MVD system of commercial performance will be operational.

Future work also includes continuing to extend the addressing modes of the MVD framebuffer to increase the image update rate toward realtime, development of the graphics processor by a third party that supports the MVD directly, continued improvements to the MOE materials and cells, and the extension of the API to incorporate tactile interaction.

From a fundamental scientific standpoint the commercial MVD systems resulting from this project will enable the visualization of 3D information in a way never before possible. From medicine to science to industry the MVD will have a sweeping impact that will accelerate our comprehension of all forms of information and enable new breakthroughs over a tremendous breadth of industries.

VI. REFERENCES

1. J. Bowren, T. Schmidt, "A New High Resolution Liquid Crystal Light Valve Projector", *SPIE Conference on Projection Displays IV*, **3296**, 105 (1998).
2. D. J. McKnight, K. M. Johnson, R. A. Serati, "256 x 256 Liquid-crystal-on-silicon spatial light modulator", *Applied Optics*, **33**, 2775 (1994).
3. M. Bone, "Optical-System Designs for LCoS Front Projectors", *Information Display*, **17**, 10 (2001).
4. I. Underwood; "VLSI design and fabrication for liquid crystal on silicon", *IEE Colloquium on "Microdisplay and smart pixel technologies"*, IEE Colloquium Digest **00/006**, 7/1-7/6 (2000).
5. D. K. Yang, L. C. Chien, J. W. Doane, "Cholesteric liquid crystal/polymer dispersion for haze-free light shutters", *Applied Physics Letters*, **60**, 3102 (1992).

VII. APPENDICES

A. MVD API Source Code

The following pages contain the source code of version 1.0 of the MVD API written in Borland Delphi 5.0.

```
unit MOEUnit;

interface

uses Forms,Windows,Dialogs,SysUtils,Math,openGL12, NiDAQUnit;

{ special types }
type

  MVD32=longword; //unsigned 32 bit
  PWord = ^Word;
  PLongword = ^LongWord;
  PByte = ^Byte;

  TMVDrec = record
    ProcessBool,
    WriteBool,
    BlockFlag:boolean;  //Whether Plane or block mode, plane assumed if not block
    GammaBool,
    BackDropBool,
    AutoScaleBool:boolean; //Auto Depth Scale enabled
    MVD_near, MVD_far: double;   //Used if not auto scaling (world space coordinates)
    MVD_NumPlanes:word;
    MVD_SizeW, MVD_SizeH: word;
    MVD_clip_near, MVD_clip_far: double; //Used if not auto scaling (z buffer space)
  end;

const
   //READ FROM IN INI OR REGISTRY OR SOMEWHERE
   MOESize_W=longword(512);
   MOESize_H=longword(512);
   MOESizeSquared=longword(MOESize_W*MOESize_H);

   // define MVD state indices
   MVDState_NumPlanes      = 1;
   MVDState_ProcessBool    = 2;
   MVDState_WriteBool      = 3;
   MVDState_GammaBool      = 4;
   MVDState_BackDropBool   = 5;
   MVDState_AutoScaleBool  = 6;
   MVDState_MVD_near       = 7;
   MVDState_MVD_far        = 8;
   MVDState_MVD_SizeW      = 9;
   MVDState_MVD_SizeH      = 10;
   MVDState_Clip_near      = 11;
   MVDState_Clip_far       = 12;
   MVDState_Blockflag      = 13;

   //define mvdError indices
   mvdError_state          = 1;
   mvdError_memory         = 2;
   mvdError_nidaq          = 3;

VAR
   MVDRec:TMVDRec; //MAKE PRIVATE AFTER DEBUG

   //ULTIMATELY PROBABLY WANT TO INTEGRATE THESE TWO
   BlockTransferBuffer2: array of MVD32;
   PlaneTransferBuffer2: array of MVD32;

   MOESizeSquared2:longword;     //2 because 2nd revision of this var

   rgbabuffer2: array of longword;  //2 because 2nd revision of this var, NB open array, (perf
 hit?)
   zbuffer2: array of Word;      //2 because 2nd revision of this var, NB open array, (perf
 hit?)

   pscale:single;
   ack2state:integer = 0;
```

```
  , req2$state:integer = 0;

    KZScale:single;
    MOEUnit_NumPlanes:word;

    MOEUnit_NearZClip:single = 0.1;
    MOEUnit_FarZClip:single = 200;

    FP_KZScale:longword; //made global 2/1/01 to allow switchoff of depth scanning

    GammaLUT:array[0..127] of byte;
    Faster_GammaLUT:array[0..255] of byte;

    maxZ:longword;
    minZ:longword;


/// MVD FUNCTION PROTOTYPES
function  mvdInit: integer;
procedure mvdClose;
procedure mvdSet_state(state_var, value:word); overload;
procedure mvdSet_state(state_var:word; value:boolean); overload;
procedure mvdSet_state(state_var:word; value:double); overload;
procedure mvdRead_state(state_var:word; var value: word); overload;
procedure mvdRead_state(state_var:word; var value: boolean); overload;
procedure mvdRead_state(state_var:word; var value: double); overload;
procedure mvdTransfer;
procedure FASTER_mvdFormat_data;
procedure mvdFormat_data;
procedure mvdClearPlaneBuffer;
procedure mvdReadGL(top, left: word); overload;
procedure mvdReadGL; overload;

implementation

////////////////////////////////////////////////////////////////////////////////

PROCEDURE SET_ACK2(state:smallint);
BEGIN
  if (state<>0) and (state<>1) then messagedlg('ACK2 state must be 0 or 1',mterror,[mbok],0);
  nidaqcall(DIG_Out_Line(1,4,3,state));
END;

FUNCTION GET_REQ2:smallint;
var state:smallint;
    bailoutcount:integer;
BEGIN
  bailoutcount:=0;
  repeat
    nidaqcall(DIG_In_Line(1,4,3,@state));
    inc(bailoutcount);
  until (state=1) or (bailoutcount > 1001);
  if bailoutcount>1001 then messagedlg('GET_REQ2 Bailed out after 1001
 iterations',mterror,[mbok],0);
  result:=state;
END;

FUNCTION GET_OFFSET_FROM_XYP(x,y,p:word):longword;
//var add,col,COLSIZE:longword;
begin
(*
  add:=0;
  COLSIZE:=16*512*NumPlanes; //the number of pixels before we hit next column on same plane
  //allow for preceding cols to go back to plane 0
  add:=add+(p*16*512);
  //calculate column & allow for preceding columns...
  col:=x div 16;
  add:=add+(col*colsize);
  //allow for rows
```

```
  add:=add+(y*16);
  //allow for pixinrow
  add:=add+(x mod 16);
  //so...
  result:=add;
  //so...
*)

 result:=(p shl 13)+(((x div 16) shl 13)*MOEUnit_NumPlanes)+(y shl 4)+(x mod 16);

 //so if...
 //x=0, y=0 p=0 result=(0 shl 13)+(((0 div 16) shl 13)*MOEUnit_NumPlanes)+(0 shl 4)+(0 mod
 16)=0
 //x=511, y=511 p=11 result=90112+3047424+8176+15=3145727
 //512^2=262144*4=1048576
end;

 //---------------------------------------------------------------------------

  ////////////////////////////////////////////////////////
 // Clear the PlaneTransferBuffer (set all values to zero //
////////////////////////////////////////////////////////
procedure mvdClearPlaneBuffer;
begin
  fillchar(PlaneTransferBuffer2,sizeof(PlaneTransferBuffer2),0);
end;

//---------------------------------------------------------------------------

//FIXME It would be better if the returned a boolean indicating a bailout condition or not. JTS
 1/2001
Procedure WaitForEndOfTransfer(QtyToTransfer,BailoutCeiling:longword);
var BailoutCount:integer;
    ulRemaining:Longint;
begin
  BailoutCount:=0;
  ulRemaining:=QtyToTransfer;
  While (ulRemaining > 0) and (BailoutCount<BailoutCeiling) do
    begin
      nidaqcall(DIG_Block_Check(1, 1, @ulRemaining));
      inc(BailoutCount);
    end;
  if BailoutCount=BailoutCeiling then messagedlg('Bailed out waiting for end of transfer after
 '+inttostr(BailoutCeiling)+'
 iterations,'+#13#10'('+inttostr(ulRemaining)+'/'+inttostr(QtyToTransfer)+'
 remaining)'+#13#10+'ie  '+inttostr(QtyToTransfer-ulRemaining)+' done',mtwarning,[mbok],0);
end;

//---------------------------------------------------------------------------


//---------------------------------------------------------------------------


//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
procedure Build_GammaLUT;
var i:integer;
begin
  for i:=0 to 127 do GammaLUT[i]:=round(127*power((i/127),3));  //quad function


  //do it twice to fill zone 128..255 which is where we'll be reading it.
  for i:=0 to 255 do Faster_GammaLUT[i]:=round(127*power(((i shr 1)/127),3));
  //mask off the upper bit here so we don't have to do it in the critical loop
  for i:=0 to 255 do Faster_GammaLUT[i]:=Faster_GammaLUT[i] and 127;
```

```
end;
//-------------------------------------------------------------------------
    {TMVDrec = record
    ProcessBool,
    WriteBool,
    GammaBool,
    BackDropBool,
    AutoScaleBool:boolean;
    MVD_near, MVD_far: double;
    MVD_NumPlanes:word;
    MVD_SizeW, MVD_SizeH: longword;
  end;}

function mvdInit: integer;
var returnU32:u32;
    returni16:i16;
begin
  {*** SET UP MVD STATE ***}
  {** Allocate RAM & Init Nidaq stuff ***}

//FIXME: VALIDATE MVD_SizeW, MVD_SizeH, Numplanes etc, errmess if not
  with MVDRec do
    begin
      if (MVD_SizeW=0) or (MVD_SizeH=0) or (MOESizeSquared=0) then
      begin
        result:=mvdError_state;
        exit;
      end;
      MOESizeSquared2:=MVD_SizeW*MVD_SizeH;

      SetLength(PlaneTransferBuffer2, MOESizeSquared);
      //messagedlg('PlaneTranferBuffer:
 '+inttostr(Length(PlaneTransferBuffer2)),mterror,[mbok],0);
      if Length(PlaneTransferBuffer2)<> MOESizeSquared then
      begin
        result:=mvdError_memory;
        exit;
      end;

      SetLength(BlockTransferBuffer2, MVD_NumPlanes*MOESizeSquared);
      //messagedlg('BlockTransferBuffer2:
 '+inttostr(Length(BlockTransferBuffer2)),mterror,[mbok],0);
      if Length(BlockTransferBuffer2)<> MVD_NumPlanes*MOESizeSquared then
      begin
        result:=mvdError_memory;
        exit;
      end;

      SetLength(rgbabuffer2, MOESizeSquared2);
      //messagedlg('rgbabuffer2: '+inttostr(Length(rgbabuffer2)),mterror,[mbok],0);
      if Length(rgbabuffer2)<> MOESizeSquared2 then
      begin
        result:=mvdError_memory;
        exit;
      end;

      SetLength(zbuffer2, MOESizeSquared2*sizeof(word));
      //messagedlg('zbuffer2: '+inttostr(Length(zbuffer2)),mterror,[mbok],0);
      if Length(zbuffer2)<> MOESizeSquared2*sizeof(word) then
      begin
        result:=mvdError_memory;
        exit;
      end;

      //FIXME: Good architecture here? hw dependance on NIDAQ?
```

```
      //get nidaq version
      returnU32:=0;
      nidaqcall(Get_NI_DAQ_Version(@returnI16));
      NIDAQversion:=inttohex(returnU32 and $FFFF,3);
      //get nidaq device name
      returni16:=23;
      returni16:=0;
      nidaqcall(Init_DA_Brds(1,@returni16));
      NIDAQdevicename:=device_name(returni16);
      //get nidaq base address
      returnU32:=0;
      nidaqcall(Get_DAQ_Device_Info(1, ND_BASE_ADDRESS,@returnU32));
      NIDAQbaseaddress:='0x'+inttohex(returnU32,8);

      //configure the groups
      nidaqcall(DIG_Grp_Config(1, 1, 4, 0, 1));
      //set up group of ports for burst handshaking, using the REQ, ACK, and PCLK signals.
      nidaqcall(DIG_Grp_Mode(1, 1, 3, 0, 0, 0, 0));
      //reverse the clock
      nidaqcall(Set_DAQ_Device_Info(1, ND_CLOCK_REVERSE_MODE_GR1, ND_OFF));
      //align the data


//nidaqcall(Align_DMA_Buffer(1,13,@BlockTransferBuffer[0],NumWordsToTransfer,NumWordsToTransfer,(
      result:=0;
  end; // with MVDRec
end;

//---------------------------------------------------------------------------

{*** DEALLOCS RAM & UNCONFIGURES NIDAQ ***}
procedure mvdClose;
begin
  PlaneTransferBuffer2:=nil;
  BlockTransferBuffer2:=nil;
  rgbabuffer2:=nil;
  zbuffer2:=nil;

  //Clear the block operation.
  DIG_Block_Clear(1, 1);
  //Unconfigure group.
  nidaqcall(DIG_Grp_Config(1, 1, 0, 0, 0));
end;

//---------------------------------------------------------------------------

procedure mvdTransfer;
var NumWordsToTransfer:longword; //number of 32 bit words to transfer
begin
  with MVDRec do
    begin
      if blockflag then
        begin
          NumWordsToTransfer := MVD_NumPlanes*MOESizeSquared2;
          // set PCLK2 to HI to set block transfer
          nidaqcall(DIG_Out_Line(1,4,1,1));
          //volume handshaking: read req2, if high then cycle ack2
          if Get_REQ2<>0 then begin Set_ACK2(1); Set_ACK2(0); end;
          //Start the pattern generation output of count "items".
          nidaqcall(DIG_Block_Out(1, 1, @BlockTransferBuffer2[0], NumWordsToTransfer));
        end else
        begin
          NumWordsToTransfer := MOESizeSquared2;
          // set PCLK2 to LO to set z-buffer transfer
          nidaqcall(DIG_Out_Line(1,4,1,0));
          //volume handshaking: read req2, if high then cycle ack2
          if Get_REQ2<>0 then begin Set_ACK2(1); Set_ACK2(0); end;
          //Start the pattern generation output of count "items".
          nidaqcall(DIG_Block_Out(1, 1, @PlaneTransferBuffer2[0], NumWordsToTransfer));
```

```
      end;
      WaitForEndOfTransfer(NumWordsToTransfer,10001);
    end; //with MVDRec
end;

//-------------------------------------------------------------------------
// MULTIPLE OVERLOADED VERSIONS OF mvdSet_state
//-------------------------------------------------------------------------
//NB Error handling should be integrated across all overloaded variants
//change these to bool returning functions
procedure mvdSet_state(state_var, value:word); overload;
begin
  with MVDRec do begin
    case state_var of
      // define MVD state indices
      MVDState_NumPlanes:     MVD_NumPlanes := value;
      MVDState_MVD_SizeW:     MVD_SizeW:= value;
      MVDState_MVD_SizeH:     MVD_SizeH:= value;
    else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
    end; // case
  end; // with MVDRec
end;

procedure mvdSet_state(state_var:word; value:boolean); overload;
begin
  with MVDRec do begin
    case state_var of
      // MVD state indices
      MVDState_ProcessBool:   ProcessBool := value;
      MVDState_WriteBool:     WriteBool := value;
      MVDState_GammaBool:     GammaBool := value;
      MVDState_BackDropBool:  BackDropBool := value;
      MVDState_AutoScaleBool: AutoScaleBool := value;
      MVDState_Blockflag:     BlockFlag := value;
    else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
   end; //case
  end; // with MVDRec
end;

procedure mvdSet_state(state_var:word; value:double); overload;
begin
  with MVDRec do begin
    case state_var of
      // define MVD state indices
      MVDState_MVD_near:      MVD_near:= value;
      MVDState_MVD_far:       MVD_far:= value;
      MVDState_Clip_near:     MVD_clip_near := value;
      MVDState_Clip_far:      MVD_clip_far := value;
      else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
    end; //case
  end; // with MVDRec
end;

//-------------------------------------------------------------------------

procedure mvdRead_state(state_var:word; var value: word); overload;
begin
//messagedlg('MVDRead_state: word',mterror,[mbok],0);
with MVDRec do begin
    case state_var of
      // define MVD state indices
      MVDState_NumPlanes:     value:=MVD_NumPlanes;
      MVDState_MVD_SizeW:     value:=MVD_SizeW;
      MVDState_MVD_SizeH:     value:=MVD_SizeH;
    else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
    end; // case
  end; // with MVDRec
end;
```

```
procedure mvdRead_state(state_var:word; var value: boolean); overload;
begin
//messagedlg('MVDRead_state: boolean',mterror,[mbok],0);
   with MVDRec do begin
    case state_var of
      // MVD state indices
      MVDState_ProcessBool :   value:=ProcessBool;
      MVDState_WriteBool :     value:=WriteBool;
      MVDState_GammaBool :     value:=GammaBool;
      MVDState_BackDropBool :  value:=BackDropBool;
      MVDState_AutoScaleBool : value:=AutoScaleBool;
      MVDState_Blockflag :     value:=BlockFlag;
    else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
   end; //case
  end; // with MVDRec
end;

procedure mvdRead_state(state_var:word; var value: double); overload;
begin
   with MVDRec do begin
    case state_var of
      // define MVD state indices
      MVDState_MVD_near :      value:=MVD_near;
      MVDState_MVD_far :       value:=MVD_far;
      MVDState_Clip_near :     value:=MVD_clip_near;
      MVDState_Clip_far :      value:=MVD_clip_far;
      else messagedlg('Problem with using MVD_set_state!!',mterror,[mbok],0);
    end; //case
  end; // with MVDRec
end;

//-------------------------------------------------------------------------


{
NEED TO SPEED UP THE GAMMA LUT,
NEED TO SPEED UP THE RGBLW SPLIT
NEED TO SPEED UP THE JJ CALCULATION
}


procedure mvdFormat_data;
var    //PROCESS PIXEL STUFF: LOCAL VARIABLES SEEM TO BE FASTER (CREATED IN CACHE?)
       Kzscale:single;
      pzbuffer:PWord;
   prgbabuffer:PLongword;
   temp:^Char;
            i:longword;
            j:longword;
            z:word;
           lw:longword;
           jj:longword;
        rgblw:longword;
        r,g,b:byte;
begin
//FIXME GHASTLY PATCHWORK STUFF HERE!
  KZscale:=1;//default
  FP_KZScale:=65536; //ie 1 in 16 bit fixed point
  minZ:=20;
  maxZ:=45;
  with MVDrec do
    begin
      if AutoScaleBool then
        begin
          //scan zbuffer, getting minz & maxz
          minZ:=65535;
          maxZ:=0;
          pzbuffer:=@zbuffer2[0];
          for i:=0 to MOESizeSquared2-1 do
```

```
          begin
            z:=pzbuffer^;      //get z value [0..65535]
            if z<65535 then    //don't include 'infinite' Z values in max min
              begin
                if z>maxZ then maxZ:=z else if z<minZ then minZ:=z;
              end;
            inc(pzbuffer);
          end;
        end; { else
        begin
          //assume minZ & maxZ are the same as the gl near/far clip planes
//
 minZ:=round((MOEUnit_FarZClip/(MOEUnit_FarZClip-MOEUnit_NearZClip))*(1-(MOEUnit_NearZClip/MVD_n
//
 maxZ:=round((MOEUnit_FarZClip/(MOEUnit_FarZClip-MOEUnit_NearZClip))*(1-(MOEUnit_NearZClip/MVD_f
          minZ:=round(MVD_clip_near*65536);
          maxZ:=round(MVD_clip_far*65536);
        end; }
      //compute scaling factor
      if MaxZ<>MinZ then KZscale:=(MVD_NumPlanes-1)/(MaxZ-MinZ);
      FP_KZScale:=round(KZscale*65536); //convert to 16 bit fixed point format
      //*** PROCESS THE DATA ****
      if ProcessBool then
        begin
          //BUILD MVD32s
          prgbabuffer:=@rgbabuffer2[0];
          pzbuffer:=@zbuffer2[0];
          for j:=0 to (MOESizeSquared2-1) do //for each pixel...
            begin
              //GET P & D
              z:=pzbuffer^;        //get z value [minZ..maxZ]
              if BackDropBool then
                begin
                  if z>maxz then z := maxz;
                end;
              z:=z-minZ;           //z [0...maxZ-minZ]
              LW:=z*FP_KZSCALE;
//            LW:=LW+65536;  //FUDGE TO FIX PLANES, adds on extra plane

              //SPLIT COLOR
              rgblw:=prgbabuffer^; //ABGR
              r:= rgblw and $FF;
              g:=(rgblw shr 8) and $FF;
              b:=(rgblw shr 16);
              //GAMMA CORRECT
              if GammaBool then
                begin
                  r:=GammaLUT[r shr 1] shl 1;
                  g:=GammaLUT[g shr 1] shl 1;
                  b:=GammaLUT[b shr 1] shl 1;
                end;
              //BUILD MVD32 STRUCTURE
              asm
                PUSH EBX
                //CALCULATE d & p
                mov   ebx,LW        //EBX=........ ..pppppp ........ ...ddddd
                mov   eax,ebx       //EAX=pppppppp pppppppp dddddddd dddddddd
                shr   eax,16        //EAX=........ ........ pppppppp pppppppp
                mov    cl,al        //CL=P
                shr    bx,11        //EBX=pppppppp pppppppp ........ ...ddddd
                //BUILD MVD32 from d,p,rgba
                shl    bx,4         // bx=                  .......d dddd....
                mov    ah,bh        //EAX=........ ........ .......d ........
                xor    bh,bh        // bx=                  ........ dddd....
                shl    bx,1         // bx=                  .......d ddd.....
                mov    al,bh        //EAX=........ ........ .......d .......d
                shl   eax,16        //EAX=.......d .......d ........ ........
                xor    bh,bh        // bx=                  ........ ddd.....
                shl    bx,1         // bx=                  .......d dd......
```

```
                mov    ax,bx        //EAX=.......d .......d .......d dd......
                 or    al,cl        //EAX=.......d .......d .......d ddpppppp
                mov   ebx,eax       //EBX=.......d .......d .......d ddpppppp

                mov   ah,r
                mov   al,g
                shl   eax,16
                mov   ah,b
                mov   al,0

                and   eax,$FEFEFE00 //EAX=rrrrrrr. ggggggg. bbbbbbb. ........
                or   eax,ebx       //EAX=rrrrrrrd gggggggd bbbbbbbd ddpppppp
                //STORE RESULT
                mov rgblw,eax
                POP EBX
              end;
              //CALCULATE PLANE BUFFER INDEX
             { jj:=(((j div 16) mod 32) shl 13)+
                   ((j div  MVD_SizeW) shl 4)+
                    (j mod 16)-1; //jj:=(col*8192)+(row*16)+pix; pp_pix-1 = FUDGE TO REMOVE
 ROGUE VERTICAL LINES}

              //CALCULATE PLANE TRANSFER BUFFER OFFSET
              asm
                    PUSH EBX
                    mov eax,J
                    shr eax,4
                    and eax,31
                    shl eax,13
                    mov ebx,J
                    shr ebx,9
                    shl ebx,4
                    add eax,ebx
                    mov ebx,J
                    and ebx,15
                    add eax,ebx
                    dec eax
                    mov JJ,eax
                    POP EBX
              end;
              PlaneTransferBuffer2[jj]:=rgblw;
              inc(prgbabuffer);
              inc(pzbuffer);
        end;//next j
    end;//processbool

    //*** WRITE THE DATA ****
   // if WriteBool then SendPlaneTransferBuffer;
   //mvdTransfer;
  end;
end;



procedure FASTER_mvdFormat_data;
var     //PROCESS PIXEL STUFF: LOCAL VARIABLES SEEM TO BE FASTER (CREATED IN CACHE?)
       Kzscale:single;
      pzbuffer:PWord;
   prgbabuffer:PLongword;
   temp:^Char;
             i:longword;
             j:longword;
             z:word;
            lw:longword;
            jj:longword;
         rgblw:longword;
         r,g,b:byte;
         p:^byte;
begin
```

```
//FIXME, GHASTLY PATCHWORK STUFF HERE!
  KZscale:=1;//default
  FP_KZScale:=65536; //ie 1 in 16 bit fixed point
  with MVDrec do
    begin
      if AutoScaleBool then
        begin
          //scan zbuffer, getting minz & maxz
          minZ:=65535;
          maxZ:=0;
          pzbuffer:=@zbuffer2[0];
          //____ CRITICAL LOOP ____________________________________________
          for i:=0 to MOESizeSquared2-1 do
            begin
              z:=pzbuffer^;      //get z value [0..65535]
              if z<65535 then    //don't include 'infinite' Z values in max min
                begin
                  if z>maxZ then maxZ:=z else if z<minZ then minZ:=z;
                end;
              inc(pzbuffer);
           end;
           //AutoScaleBool:=false;
          //____ END CRITICAL LOOP ________________________________________
        end; { else
        begin
          //assume minZ & maxZ are the same as the gl near/far clip planes
//
 minZ:=round((MOEUnit_FarZClip/(MOEUnit_FarZClip-MOEUnit_NearZClip))*(1-(MOEUnit_NearZClip/MVD_n
//
 maxZ:=round((MOEUnit_FarZClip/(MOEUnit_FarZClip-MOEUnit_NearZClip))*(1-(MOEUnit_NearZClip/MVD_f
          minZ:=round(MVD_clip_near*65536);
          maxZ:=round(MVD_clip_far*65536);
        end; }

      //compute scaling factor
      if MaxZ<>MinZ then KZscale:=(MVD_NumPlanes-1)/(MaxZ-MinZ);
      FP_KZScale:=round(KZscale*65536); //convert to 16 bit fixed point format
      //*** PROCESS THE DATA ****
      if ProcessBool then
        begin
          //BUILD MVD32s
          prgbabuffer:=@rgbabuffer2[0];
          pzbuffer:=@zbuffer2[0];
//_________ CRITICAL LOOP ____________________________________________
          for j:=0 to (MOESizeSquared2-1) do //for each pixel...
            begin
              //GET P & D
              z:=pzbuffer^;       //get z value [minZ..maxZ]
//if BackDropBool then if z>maxz then z := maxz; MOVE THIS OUTSIDE!
              z:=z-minZ;          //z [0...maxZ-minZ]
              LW:=z*FP_KZSCALE;

              rgblw:=prgbabuffer^; //ABGR

              //GAMMA CORRECT EACH COMPONENT
                  asm
                    //SPLIT RGBLW TO RGB
                    mov eax,rgblw  //ABGR
                    mov R,al
                    mov G,ah
                    shr eax,8
                    mov B,ah
                  end;
                    r:=Faster_GammaLUT[r];
                    g:=Faster_GammaLUT[g];
                    b:=Faster_GammaLUT[b];
                  asm
                    //REASSEMBLE RGBLW
                    mov dh,R    //EDX=..... ..... .rrrrrrr .....
```

```
                    mov dl,G    //EDX=..... ..... .rrrrrrr .ggggggg
                    shl edx,16  //EDX=.rrrrrrr .ggggggg ..... .....
                    mov dh,B    //EDX=.rrrrrrr .ggggggg .bbbbbbb .....
                    shl edx,1   //EDX=rrrrrrr. ggggggg. bbbbbbb. .....
                    //BUILD MVD32 STRUCTURE
                    PUSH EBX
//*************************************************************************
//* THIS NEXT BIT (TO THE POP) IS SUB OPTIMAL DUE TO PARTIAL PROCESSOR STALLS  *
//* CAUSED PRIMARILY BY MIXING 8 & 32 BIT OPERANDS - SEE GAMASUTRA            *
//*************************************************************************
                    //ISOLATE P FROM LW
                    mov ebx,LW  //EBX=..... ..pppppp DDDDDddd .....
                    shr ebx,8   //EBX=..... ..... ..pppppp DDDDDddd
                    mov cl,bh   //CL=P at this point
                    //SPLIT UP D APPROPRIATELY
                    xor bh,bh   //EBX=..... ..... ..... DDDDDddd
                    shl bx,1    //EBX=..... ..... .....D DDDDddd.
                    mov bh,ah   //EAX=..... ..... .....D .....
                    xor bh,bh   //EBX=..... ..... ..... DDDDddd.
                    shl bx,1    //EBX=..... ..... .....D DDDdd..
                    mov bh,al   //EAX=..... ..... .....D .....D
                    shl eax,16  //EAX=.....D .....D ..... .....
                    xor bh,bh   //EBX=..... ..... ..... DDDdd..
                    shl bx,1    //EBX=..... ..... .....D DDdd...
                    and bx,192  //EBX=..... ..... .....D DD.....
                    mov ax,bx   //EAX=.....D .....D .....D DD.....
                    //BUILD MVD32 from d,p,rgba
                    or  al,cl   //EAX=.....d .....d .....d ddpppppp
                    or  eax,edx //EAX=rrrrrrrd ggggggd bbbbbbbd ddpppppp
                    //STORE MVD32 RESULT
                    mov RGBLW,eax
                    //CALCULATE PLANE TRANSFER BUFFER OFFSET
                    mov eax,J
                    shr eax,4
                    and eax,31
                    shl eax,13
                    mov ebx,J
                    shr ebx,9
                    shl ebx,4
                    add eax,ebx
                    mov ebx,J
                    and ebx,15
                    add eax,ebx
                    dec eax
                    mov JJ,eax
                    POP EBX
                end;
            PlaneTransferBuffer2[jj]:=rgblw;
            inc(prgbabuffer);
            inc(pzbuffer);
        end;//next j
    end;//processbool
  end;
end;




//---------------------------------------------------------------------------
procedure mvdReadGL; overload;
begin
  with mvdrec do
  begin
    glreadpixels(0, 0, MVD_SizeW, MVD_SizeH,          GL_RGBA, GL_UNSIGNED_BYTE,
 @rgbabuffer2[0]);
```

```
    glReadPixels(0, 0, MVD_SizeW, MVD_SizeH, GL_DEPTH_COMPONENT, GL_UNSIGNED_SHORT,
 @zbuffer2[0]);
  end;
end;

procedure mvdReadGL(top, left:word); overload;
begin
  with mvdrec do
  begin
        glreadpixels(top, left, MVD_SizeW, MVD_SizeH,          GL_RGBA, GL_UNSIGNED_BYTE,
 @rgbabuffer2[0]);
        glReadPixels(top, left, MVD_SizeW, MVD_SizeH, GL_DEPTH_COMPONENT, GL_UNSIGNED_SHORT,
 @zbuffer2[0]);
  end;
end;

//SELF INITIALIZING CODE
begin
  MOEUnit_NumPlanes:=12; //SHOULD BE READ FROM INI OR SOMEWHERE
  //MOEUnit_NumPlanes:=MVD_NumPlanes;
  Build_GammaLUT;
end.



(*

DATA TRANSFER MODES
===================
1) BLOCK MODE
The entire 50 plane data set is sent at 32 bits per pixel:
[R7x G7x B7x x8].
AA is done by host computer if desired.
NB 11 bits are {redundant/not used/reserved for future development}.

2) PLANE (Z BUFFER) MODE
A single page of 512*512 pixels is sent, along with D & P as a 32 bit word:
[R7D G7D B7D D2P6]
The MOE uses P ro place RGB in the correct address and uses D to modulate.

DISPLAY RAM
===========
The display RAM in the MOE3 is laid out as follows:

There are 50 planes:
Each plane is 512*512 pixels
Each plane is arranged as 32 columns, each 16 pixels wide and 512 rows deep
Memory locations are sequential as follows:

for column:=1 to 32 do
  begin
    for plane:=1 to 50 do
      begin
        for row:=1 to 512 do
          begin
            for pixel:=1 to 16 do
              begin
              end;
          end;
      end;
  end;

This equates to (32*16)*512*50= 13,107,200 pixels

Each pixel is 32 bit and contains the following data:

rrrrrrrd ggggggd bbbbbbd ddpppppp
```

The entire structure is therefore:
*13107200*4= 52,428,800 bytes,*
52428800/1024= 51,200 kB
51200/1024= 50 Mb

This needs to be dumped to the MOE at 30fps, a transfer rate of 1500MB/sec or 1.46GB/sec

So the MOE driver program needs to:
a) Read the Color Buffer
b) Read the Depth Buffer
PAGE MODE:
c) Compute Plane and Delta data for each pixel
*d) Reformat the data into the 512*512*[DR7 DG7 DB7 D2P6] array structure*
*e) Dump the 512*512*32bit structure to the MOE via PCI bus*
BLOCK MODE:
c) Compute Plane and Delta data for each pixel
*d) Populate the entire 512*512*50*[xR7xG7xB7] structure based on RGB & Plane*
d) Modulate RGB on a per pixel basis based on the delta
*e) Dump the 512*512*50*24bit structure to the MOE via PCI bus*

a) Read the Color Buffer

This is done using glReadPixels, a non hw accelerated function (unless using certain models of Intergraph PCs). Ultimately we want to express each color as a 7 bit number, so ideally we'd like glReadPixels to return a byte between 0 & 127. Happily OpenGL does this for us if we specify a format of GL_BYTE in the glReadPixels command.

*glReadPixels(0,0,512,512,GL_BYTE,GL_RGB,@pixels) will give us a 512*512 array of RGB triplet bytes. The total size of the array will be 512*512*3=786,432 bytes=768kB or 0.75MB and we can rely on the msb in each byte being = 0.*

RGBpixels:array[0..786432] of byte;

value: 0rrrrrrr 0ggggggg 0bbbbbbb 0rrrrrrr 0ggggggg 0bbbbbbb
pixel: 0------------------------ 1------------------------
byte: 0------- 1------- 2------- 3------- 4------- 5-------

b) Read the Depth Buffer

This is also done using glReadPixels. Ultimately we want to express the depth as a 5 bit number representing the plane [0..50], and a 5 bit number representing the delta. At the moment (MOE2) we read this as a 4 byte [0..1] glfloat because we need the fractional part to calculate the delta.
*This is quite slow because we're reading 512*512*4=1048576= 1MB.*
It also provides us with more data than we can use, since we only have 5 bit delta. Because we have only 32 delta values we only need fractional accuracy to 1/32=0.03125.

*The usable range for us is therefore 50*32=1600. 1600 is 11001000000 in binary, ie our usable range will fit into 11 bits, so reading a 16 bit value will be more than adequate for our needs.*

*glReadPixels(0,0,512,512,GL_SIGNED_SHORT,GL_DEPTH_COMPONENT,@depthpixels) will give us a 512*512*2 array of words. The total size of the array will be 512*512*2=524,288 bytes=512kB or 0.5MB.*

Although this in theory should execute twice as fast as a float read, we have some post processing to do. Since we're reading signed shorts the range is 0..32768 and we can rely on the msb=0.

The range of each depthpixel is [0..32767]. From this integer we have to derive a float number in the range [0..49]. pscale=49/32767=0.0014954069643.
So we could multiply each depth pixel by this number.
This is an awkward number to try to optimize, but at least its a constant...

c) Compute Plane and Delta data for each pixel

```
-----------------------------------------

plane=depthpixel*pscale;
delta=frac(plane)
plane=round(plane)

so..

MOE2 float read....
depthpixel=0.555
plane:=50*depth=27.75
delta:=frac(plane)=0.75
plane:=round(plane)=27

MOE3 word read....
depthpixel=0.555*32768=18186
plane=18186*pscale=27.749633789
delta=frac(27.749633789)=0.749633789
plane=round(27.749633789)=27

so the error is 27.75-27.749633789=0.000366211, not worth worrying about.

In order to pack the delta into 5 bits for the output 32 bit word we take
 delta:=round(delta*31)??

d) Reformat the data into the new array structure

rgbpixels   depthpixels
    |          |      |
    |       plane  delta
    |          |      |
    ====MOE===========

TRGBtriplet=array[0..2] of byte;
RGBpixels:array[0..262144] of TRGBtriplet;
depthpixels:array[0..262144] of integer;
MOE:array[0..31][0..49][0..511][0..15] of longword

pixelnum:=0;
for column:=1 to 32 do
  begin
    for plane:=1 to 50 do
      begin
        for row:=1 to 512 do
          begin
            for pixel:=1 to 16 do
              begin
                r:=RGBpixels[pixelnum][0];
                g:=RGBpixels[pixelnum][1];
                b:=RGBpixels[pixelnum][2];
                f:=DepthPixel[pixelnum]*pscale;  // ugh, floating point
                p:=round(f);                              // ugh, floating point
                d:=round((f-p)*31);                       // ugh, floating point
                //BIT TWIDDLING STARTS HERE
                lw:=(r shl 1)+(d and 1);              //lw=00000000 00000000 00000000 rrrrrrrd
                lw:=lw shl 8;
                lw:=lw+(g shl 1)+((d and 2) shr 1);   //lw=00000000 00000000 rrrrrrrd gggggggd
                lw:=lw shl 8;
                lw:=lw+(b shl 1)+((d and 4) shr 2);   //lw=00000000 rrrrrrrd gggggggd bbbbbbbd
                lw:=lw shl 8;
                lw:=lw+((d and 24) shl 3)+(p and 63); //lw=rrrrrrrd gggggggd bbbbbbbd ddpppppp
                MOE[column][plane][row][pixel]:=lw;
                inc(pixelnum);
              end;
          end;
      end;
  end;

Of course it would be nice to loop this in machine code...
```

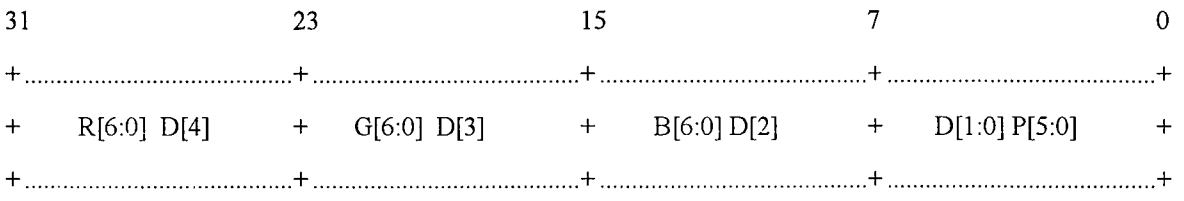
The bit twiddling is straightforward, the multiplication by an fp constant should be an FMUL (precludes the use of SIMD - careful) and the input addressing all has the same offset. Calculating the output address might be a pain....

e) Dump the structure to the MOE via PCI bus
Just pass NIDAQ the address of the MOE struct and hit the go button.

**)*

B. MVD322 Data Word Format

The data word assembled by the *mvdFormat_data* API function is known as an MVD32 type. It is a 32-bit data word with the following format:



Where:

R[6:0] is the 7-bit red data,

G[6:0] is the 7-bit green data,

B[6:0] is the 7-bit blue data,

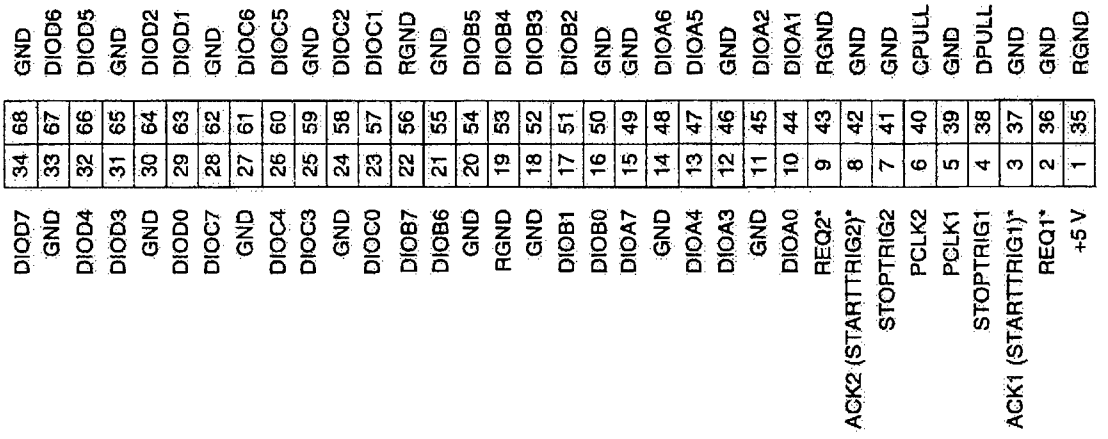
D[4:0] is the 5-bit fractional portion of the depth, and

P[5:0] is the 6-bit integer portion of the pixel depth

This data word format was developed to maximize the rate at which the API function *mvdFormat_data* runs. The format of the color data retrieved from the framebuffer by *mvdReadgl* is a 32-bit number with the red, green and blue data already in the correct locations within the MVD32 data word. The only operation required is to splice the pieces of the depth value into the data word.

C. Data Transfer Cable Pin Assignments

The following figure is scanned from the documentation of the National Instruments PCI-DIO-32HS high speed data transfer card. It is Figure 4-1 indicating the pin assignments of the 68-pin output connector. (July 1997 Edition)



Pins DIOD[7:1] correspond to R[6:0] in the MVD32 data word

Pin DIOD[0] correspond to D[4] in the MVD32 data word

Pins DIOC[7:1] correspond to RG6:0] in the MVD32 data word

Pin DIOC[0] correspond to D[3] in the MVD32 data word

Pins DIOB[7:1] correspond to G[6:0] in the MVD32 data word

Pin DIOB[0] correspond to D[2] in the MVD32 data word

Pins DIOA[7:6] correspond to D[1:0] in the MVD32 data word

Pins DIOA[5:0] correspond to P[5:0] in the MVD32 data word

Pin PCLK1 is the data clock input from the MVD framebuffer

Pins ACK1, and REQ1 are the data handshaking pins. The ACK1 pin is asserted when the card is ready to send data, and the REQ1 signal is asserted when the framebuffer is ready to receive data. Both signal must be asserted for data transfer to occur.

Pin PCLK2 is an output pin that signals the data transfer mode: HIGH indicates plane transfer mode, LOW indicates block transfer mode. This signal is only detected by the framebuffer between data transfers.

Pins ACK2 and REQ2 are the volume handshaking signals. The framebuffer asserts REQ2 when it is ready to receive a new volume of data in either plane transfer mode or block transfer mode. The mvdTransfer API function checks for a HIGH condition on REQ2 before beginning a transfer. If a HIGH condition is found, the function toggles

ACK2 first HIGH then LOW to tell the MVD framebuffer that transfer is about to begin. Transfer follows immediately.

The rest of the pins are either ground (GND) as indicated or have no MVD function.

D. SGI Device Driver source code

The following pages contain the source code an SGI IRIX 6.2 device driver for the National Instruments PCI-DIO-32HS high speed data transfer card.

```
/*****************************************************************************
 *****          P C I - D I O - 3 2 H S   P C I Device Driver             *****
 *****                                                                    *****
 *****************************************************************************
*/
#include <sys/types.h>
#include "sys/cmn_err.h"
#include "sys/sema.h"
#include <sys/param.h>
#include <sys/errno.h>
#include <sys/syslog.h>
#include <sys/conf.h>
#include <sys/pio.h>
#include "sys/systm.h"
#include <sys/time.h>
#include <sys/ksynch.h>
#include <sys/ktime.h>
#include <sys/invent.h>
#include <sys/kmem.h>
#include <sys/kabi.h>
#include <sys/mload.h>
#include <sys/ddi.h>
#include <sys/cred.h>
#include <sys/immu.h>
#include <sys/region.h>
#include <sys/alenlist.h>
#include <sys/ioerror.h>
#include <sys/PCI/PCI_defs.h>
#include <sys/PCI/pciio.h>


#include "diodrvr.h"
#include "diodrvr_user.h"

/*  ddi/dki and irix required  */
char    *diodrvr_mversion = M_VERSION;   /*  loadable driver requirement */
int      diodrvr_devflag = D_MP;          /*  ddi/dki requirement         */

/* =====================================
 *    Device Driver/PCI entry routines
 * ===================================== */
int diodrvr_unload(void);
int diodrvr_open(dev_t *, int, int, cred_t *);
int diodrvr_close(dev_t, int, int, cred_t *);
int diodrvr_read( dev_t, uio_t *, cred_t *);
int diodrvr_write( dev_t, uio_t *, cred_t *);
int diodrvr_ioctl(dev_t, int, void *, int, cred_t *, int *);
int diodrvr_map ( dev_t, vhandl_t *, off_t, size_t, uint_t );
int diodrvr_unmap ( dev_t, vhandl_t * );
int diodrvr_init();
int diodrvr_reg(void);
int diodrvr_reg(void);

int diodrvr_attach(vertex_hdl_t);
int diodrvr_detach(vertex_hdl_t);

/* =====================================
 *       Helper functions
 * ===================================== */
void            diodrvr_reset(card_t *);
void            diodrvr_burst_clkout(card_t *, int *);
void            diodrvr_burst_clkin(card_t *, int *);
void            diodrvr_dma_group1(card_t *, int *);
static int      diodrvr_Strategy ( buf_t  * );
static          diodrvr_dmapage_t * diodrvrMakeChain ( card_t *, alenlist_t, int);
static void diodrvrStartProgDma ( card_t *, iopaddr_t, int, int);
static void diodrvrStartSingleDma ( card_t *, alenaddr_t, size_t, int );
static void diodrvrTimeOut ( card_t * );
void            diodrvr_int_countexp(card_t * );
```

```
void            Read_Important_Registers(card_t *);
void            diodrvr_run(card_t * );
void            diodrvr_stop(card_t * );
static void diodrvrReportTime ( caddr_t, card_t *, int );
static void diodrvrDiffTime ( struct timespec *, struct timespec *, struct timespec * );
static void diodrvrConvTime ( struct timespec *, diodrvr_timeval_t * );
static void     diodrvrUserTime ( card_t *, int * );

/*
 *      Our error handler and interrupt handler
 *      Note: We do not set any error handler
 */
static error_handler_f diodrvr_error;
void diodrvr_intr( intr_arg_t );

int             dmaChannel = 1;
int             drqnum = 1;
uchar_t         *DMA_BASE_ADDR;

/*
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~                                                    ~~~~~~~~~~~~~
~~~~~~~~~~~    D r i v e r ' s   E n t r y   R o u t i n e s   ~~~~~~~~~~~~~~
~~~~~~~~~~~                                                    ~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

/*************************************************************************
 ***                  d i o d r v r _  i n i t                          ***
 *************************************************************************
 *
 *  Name:       diodrvr_init
 *
 *  Purpose:    Called by kernel. For Irix6.4 only: We do not register
 *              here but we will do it in diodrvr_reg. For init, we simply
 *              do nothing.
 *
 *  Returns:    None
 *
 *************************************************************************/
int
diodrvr_init()
{
        return(0);
}

/*************************************************************************
 ***                  d i o d r v r _  r e g                           ***
 *************************************************************************
 *
 *  Name:       diodrvr_reg
 *
 *  Purpose:    Called by kernel when the driver is loaded.
 *              Here we register ourselves for the card, identified by
 *              Vendor and Device ID.
 *
 *  Note:       This is for Irix6.4 only.
 *
 *  Returns:    None
 *
 *************************************************************************/
int
diodrvr_reg()
{
    register int  ret;

    /*    Register and identify the card   */
    ret = pciio_driver_register(VENDOR_ID, DEVICE_ID, DRIVER_PREFIX, 0);
```

```
    if ( ret ) {
        cmn_err (CE_WARN, "diodrvr_reg: registration returned %d" , ret );
    }

    return(ret);
}

/*************************************************************************
 ***                  d i o d r v r _ a t t a c h                      ***
 *************************************************************************
 *
 *  Name:       diodrvr_attach
 *
 *  Purpose:    Called by the kernel. Our card is installed and hence
 *              we are called. Prepare everything necessary to handle
 *              the card. Note that when the card is found, the following
 *              is set in the Command field of Config space:
 *                      - Bus master enabled/
 *                      - Memory and IO space access enabled.
 *                      - Cache line is set to 0x20 (32)
 *                      - Latency Timer is set to 0x30 (48)
 *
 *              For Chameleon, beside Configuration address space, we need 4
 *              Memory address spaces to be mapped:
 *                 Base_Reg 0 = MITE Registers and Fifos
 *                 Base_Reg 3 = Normal DMA registers
 *                 Base_Reg 4 = Configuration Register.
 *
 *  Returns:    0 for Success, or errno
 *
 *************************************************************************/
int
diodrvr_attach(vertex_hdl_t conn)
{

    card_t                      *cp;
    caddr_t                     cfg_adr, mem_ptr, mite_adr, dio_adr, norm_adr;
    pciio_piomap_t      cfg_map, mite_map, dio_map, norm_map;
    register int        ret, i;
    uint_t                      vendor_id, device_id, base_reg, cmd_reg, tmp_int;
    vertex_hdl_t        diodrvr_vhdl;
    device_desc_t       dev_desc;
    long                        dev_win_data;
    long                        testlong;
    char                        testchar;
        register uint_t LCR2bits, MGARbits, PLIVRbits;
        register uint_t blklim, cacheLineWords;

    /* dev_desc = device_desc_default_get(conn); */
    dev_desc    = 0;

        pciio_priority_set(conn, PCI_PRIO_HIGH);

    /* =========================
     *  Configuration Space
     * ========================= */

    cfg_map = (pciio_piomap_t)NULL;
    cfg_adr = (caddr_t)pciio_pio_addr (
                conn,                                   /*  connection vertex      */
                dev_desc,                               /*  default device descr.  */
                PCIIO_SPACE_CFG,        /* dioig space wanted     */
                0,                                              /* from the start of space */
                DIODRVR_CONFIG_HDR, /* for this many bytes     */
                &cfg_map,                               /* in case we need piomap  */
                0);                                             /* unused flag             */

    if ( cfg_adr == (caddr_t)NULL ) {
        if ( cfg_map )  pciio_piomap_done ( cfg_map );
```

```
                cmn_err ( CE_WARN, "diodrvr_attach: Cannot get to Config space" );
    return(EIO);
}


/* ==============================
 *   Configuration data printout
 * ============================== */

/*  vendor_id, device_id  */
tmp_int   = Inp32(cfg_adr);
vendor_id = tmp_int & 0x0000ffff;
device_id = tmp_int >> 16;

#ifdef DEBUG
printf ("diodrvr_attach:  Config values\n");
printf ("vendor_id = 0x%x, device_id = 0x%x\n", vendor_id, device_id );

/*  command and status  */
tmp_int = Inp32(cfg_adr + 0x04);
printf ("Command   = 0x%x, Status    = 0x%x\n", (tmp_int & 0x0000ffff),
        (tmp_int >> 16) );

/*  interrupt line and pin   */
tmp_int = Inp32(cfg_adr + 0x3C);
printf ("Int Line = %d, Int Pin = %d\n", (tmp_int & 0x000000ff),
           (tmp_int & 0x0000ff00) >> 8 );

/*  set the cache line size and latency timer  */
tmp_int = Inp32(cfg_adr + 0x0C);
    tmp_int = (tmp_int & 0xFFFF0000) | 0xFF20;
    Out32(cfg_adr + 0x0C, tmp_int);


    /*  cache line size and latency timer  */
tmp_int = Inp32(cfg_adr + 0x0C);
printf ("Cache line size = %d,  latency timer = %d\n",
        (tmp_int & 0x000000ff), (tmp_int & 0x0000ff00) >> 8 );

/*  all 6 base registers  */
for ( i = 0; i < 6; i++ ) {
      printf ("Base_Reg_%d = 0x%x\n", i, Inp32(cfg_adr+0x10+(i*4)) );
}

    dev_win_data = (Inp32(cfg_adr+0x14) & 0xffffff00) | 0x80;
    printf("dev_win_data: 0x%x\n", dev_win_data);

#endif

/* ==========================
 *       MITE Registers
 * ========================== */

/*  Get MITE Register addresses */
mite_map = (pciio_piomap_t)NULL;
mite_adr = (caddr_t)pciio_pio_addr (
            conn,                    /* connection vertext      */
                          dev_desc,          /* default device descr.   */
            PCIIO_SPACE_WIN(0), /* Base register 0 space   */
            0,                  /* from the start of space */
            MITE_RAM_SIZE,      /* for this many bytes     */
            &mite_map,          /* in case we need piomap  */
            0);                 /* unused flag             */

if ( mite_adr == (caddr_t)NULL ) {
    cmn_err(CE_WARN, "diodrvr_attach: Cannot get to MITE address space" );
    if ( cfg_map )  pciio_piomap_done ( cfg_map );
    if ( mite_map ) pciio_piomap_done ( mite_map );
    return (EIO);
```

```
}

    /* write the DIO devices address in the MITE */
    Out32(mite_adr + MITE_DEV_WIN, dev_win_data);

    /* set max retries to infinite
    MGARbits = Inp32(mite_adr + MITE_MGAR);
    MGARbits = MGARbits | (MITE_MGAR_DISABLEMAX);
    Out32(mite_adr + MITE_MGAR, MGARbits); */

    /* match the MITE cache width to the processor cache width
    PLIVRbits = Inp32(mite_adr + MITE_PLIVR);
    cacheLineWords = (PLIVRbits & 0x000000FF)/4;

    switch(cacheLineWords) {
            case 2:
                    blklim = MITE_LCR2_BLKLIM_2;
                    break;
            case 4:
                    blklim = MITE_LCR2_BLKLIM_4;
                    break;
            case 8:
                    blklim = MITE_LCR2_BLKLIM_8;
                    break;
            case 16:
                    blklim = MITE_LCR2_BLKLIM_16;
                    break;
            default:
                    blklim = 0xFFFFFFFF;
                    break;  */
    /*}
    if (blklim != 0xFFFFFFFF) { */

            blklim = MITE_LCR2_BLKLIM_8;
            LCR2bits = Inp32(mite_adr + MITE_LCR2);
            /* mask out old values */
            LCR2bits &= ~(MITE_LCR2_BLKLIM | MITE_LCR2_BLKOBND);
            /* now write new values */
            LCR2bits |= (blklim | MITE_LCR2_BLKOBND);
            Out32(mite_adr + MITE_LCR2, LCR2bits);

    /*}*/

    DMA_BASE_ADDR = (dmaChannel * 0x0100L) + (uchar_t *)mite_adr + 0x500;

/* ================================
 *     DIO Registers
 * ================================ */
dio_map = (pciio_piomap_t)NULL;
dio_adr = (caddr_t)pciio_pio_addr (
            conn,                           /* connection vertext      */
                        dev_desc,           /* default device descr.   */
            PCIIO_SPACE_WIN(1), /* Base register 1 space   */
            0,                  /* from the start of space */
            DIO_RAM_SIZE,       /* for this many bytes     */
            &dio_map,           /* in case we need piomap  */
            0);                 /* unused flag             */


if ( dio_adr == (caddr_t)NULL ) {
    cmn_err(CE_WARN, "diodrvr_attach: Cannot get to Config Register" );
    if ( cfg_map )  pciio_piomap_done ( cfg_map );
    if ( mite_map ) pciio_piomap_done ( mite_map );
    if ( dio_map ) pciio_piomap_done ( dio_map );
    return (EIO);
}

    /* Test the card by reading the test registers */
    printf("dio test data: 0x%x\n", Inp32(dio_adr+24));
```

```
    /*  allocate an internal structure for this card and save everything */
    cp = (card_t *)kmem_zalloc ( sizeof(card_t), KM_NOSLEEP );
    if ( cp == (card_t *)NULL ) {
        cmn_err(CE_WARN, "diodrvr_attach: Cannot allocate memory" );
        if ( cfg_map )  pciio_piomap_done ( cfg_map );
        if ( mite_map ) pciio_piomap_done ( mite_map );
        if ( dio_map ) pciio_piomap_done ( dio_map );
        return(ENOMEM);
    }

    #ifdef DEBUG
    printf ("diodrvr_attach: mite_adr = 0x%x,     dio_adr = 0x%x\n" ,
            mite_adr, dio_adr );
    printf ("             cfg_adr  = 0x%x\n",
            cfg_adr);

    printf ("Maps allocated: %s %s %s\n",
            mite_map ? "MITE":"",
            dio_map ? "dio":"",
            cfg_map  ? "Cfg":"" );
    #endif

    cp->conn     = conn;
    cp->cfg_adr  = cfg_adr;
    cp->mite_adr = mite_adr;
    cp->dio_adr  = dio_adr;

    cp->mite_map = mite_map;

    cp->dio_map  = dio_map;
    cp->cfg_map  = cfg_map;
    cp->kernel_abi = get_current_abi();  /* ABI_IRIX5_64 for 64-bit kernel */

        /* reset the card */
        diodrvr_reset( cp );

        /* Test the card by writing to the output registers */
        printf("attempting to test output channels\n" );
        Out32(dio_adr+32, 0);
        Out32(dio_adr+36, 0);
        Out32(dio_adr+32, 0xffffffff);

        testlong = 0xffffffff;
        testchar = 0xff;
/*
        for(i=0;i<21;i++)
        {
                Out32(dio_adr+28, testlong);
                testlong = testlong ^ 0xffffffff;
                testchar = testchar ^ 0xff;
        }

*/
    /*  initialize our mutex lock and sv_t for sleep/wake  */
    DIODRVR_LOCK_INIT(&cp->diodrvr_mlock);
    SV_INIT(&cp->diodrvr_sv, SV_DEFAULT, "diodrvrsv" );

    /*    create a vertex for our device off of the connection point */
    ret = hwgraph_char_device_add(conn, "diodrvr", "diodrvr_", &diodrvr_vhdl );
    if ( ret != GRAPH_SUCCESS ) {
        cmn_err (CE_WARN, "diodrvr_attach: could not addr vertex" );
        if ( cfg_map )  pciio_piomap_done ( cfg_map );
        if ( mite_map ) pciio_piomap_done ( mite_map );
        if ( dio_map ) pciio_piomap_done ( dio_map );

        kmem_free ( cp, sizeof(card_t) );
        return(EIO);
    }
```

```
    hwgraph_chmod( diodrvr_vhdl, 0666 );
    cp->vhdl      = diodrvr_vhdl;

    /*  once iodioig and DIODRVR_MKHWGRAPH works, remove this area  */
    ret =  hwgraph_edge_add ( hwgraph_root, diodrvr_vhdl, "diodrvr" );
    if ( ret != GRAPH_SUCCESS ) {
            cmn_err (CE_WARN, "diodrvr_attach: Cannot create node /hw/diodrvr" );
    }

    /* ====================================
     *     Interrupt Handler Registration
     * ==================================== */
    cp->dev_intr = pciio_intr_alloc ( conn, dev_desc,
                                      PCIIO_INTR_LINE_A,
                                      diodrvr_vhdl );

    if (cp->dev_intr == (pciio_intr_t)NULL){
                cmn_err(CE_WARN, "diodrvr_attach: Can't pciio_intr_alloc" );
        if ( cfg_map )  pciio_piomap_done ( cfg_map );
        if ( mite_map ) pciio_piomap_done ( mite_map );
        if ( dio_map ) pciio_piomap_done ( dio_map );

        kmem_free ( cp, sizeof(card_t) );
        hwgraph_edge_remove(conn, "diodrvr", &diodrvr_vhdl);
        device_info_set ( diodrvr_vhdl, 0 );
        hwgraph_vertex_destroy ( diodrvr_vhdl );
                return(EIO);
    } /* if can't allocate interrupt handler */

    ret = pciio_intr_connect ( cp->dev_intr, (intr_func_t)diodrvr_intr,
                               (intr_arg_t)cp, 0 );

    if ( ret != 0 ) {
        cmn_err(CE_WARN, "diodrvr_attach: Cannot register interrupt handler" );
        if ( cfg_map )  pciio_piomap_done ( cfg_map );
        if ( mite_map ) pciio_piomap_done ( mite_map );
        if ( dio_map ) pciio_piomap_done ( dio_map );

                kmem_free ( cp, sizeof(card_t) );
        hwgraph_edge_remove(conn, "diodrvr", &diodrvr_vhdl);
        device_info_set ( diodrvr_vhdl, 0 );
        hwgraph_vertex_destroy ( diodrvr_vhdl );
        return (EIO);
    } /* if can't connect to interrupt line */

   /* ========================================
    *    Register an error handler for 6.4
    * ======================================== */

#if 0
    pciio_error_register(conn, (error_handler_f *)diodrvr_error, cp );
#endif

    cp->status    = CARD_ATTACHED;

    /*   allocate memory for mapping
    for ( i = MAP_PAGES; i > 0; i-- ) {
         cp->mappedkv = kmem_alloc (i * NBPP,
                                    KM_NOSLEEP | KM_PHYSCONTIG | KM_CACHEALIGN);
        if ( cp->mappedkv != (caddr_t)NULL )
                break;
    }
    if ( cp->mappedkv == (caddr_t)NULL ) {
            cmn_err (CE_NOTE, "diodrvr_attach: Not enough memory for mapping");
            cmn_err (CE_WARN, "diodrvr_attach: No mapping is allowed");
    }
    else {
        cp->mappedkvlen = i * NBPP;
```

```
        cmn_err (CE_NOTE, "diodrvr_attach: %d bytes allocated, %d available for mapping",
                cp->mappedkvlen, cp->mappedkvlen - LESS_MAP );
    } */

    /*  save our structure  */
    device_info_set ( diodrvr_vhdl, (void *)cp );

    cmn_err ( CE_NOTE, "diodrvr_attach: driver is ready" );
    return(0);
}

/*****************************************************************************
 ***                    d i o d r v r _ d e t a c h                        ***
 *****************************************************************************
 *
 *  Name:       diodrvr_detach
 *
 *  Purpose:    Detaches a driver. Called by the pciio infrastructure
 *              once for each vertext representing a crosstalk widget
 *              when unregistering the driver.
 *
 *  Returns:    0 Success or errno
 *
 *****************************************************************************/

int
diodrvr_detach(vertex_hdl_t conn)
{
    register card_t       *cp;
    vertex_hdl_t           vhdl;

    if ( hwgraph_traverse(conn, "diodrvr", &vhdl) != GRAPH_SUCCESS )
        return(-1);

    hwgraph_vertex_unref(vhdl);   /* update the ref count for above call */
    cmn_err(CE_NOTE,
           "diodrvr_detach: Unregister Interrupt handler and free up mem" );

    cp = (card_t *)device_info_get ( vhdl );

    /*  our struct is there ? clean up our driver's stuff */
    if ( cp != (card_t *)NULL ) {

                /*  free up memory for mapping   */
                if ( cp->mappedkv )
                        kmem_free ( cp->mappedkv, cp->mappedkvlen );

                /*  Unregister the Interrupt Handler  */
                pciio_intr_disconnect(cp->dev_intr);
                pciio_intr_free(cp->dev_intr);

                #if 0
                        /*  unregister error handler  */
                pciio_error_register(conn, 0, 0);
                #endif

                /*  free up all our maps  */
                if ( cp->cfg_map )  pciio_piomap_done ( cp->cfg_map );
                if ( cp->mite_map ) pciio_piomap_done ( cp->mite_map );
                if ( cp->dio_map ) pciio_piomap_done ( cp->dio_map );

                /*  free up the struct itself  */
                kmem_free ( cp, sizeof(card_t) );

    }

    /*  free up our vertextes   */
    hwgraph_edge_remove ( hwgraph_root, "diodrvr", NULL );
    hwgraph_edge_remove(conn, "diodrvr", NULL );
```

```
    device_info_set ( vhdl, 0 );
    hwgraph_vertex_unref( vhdl );
    hwgraph_vertex_destroy ( vhdl );

    return(0);
}

/*************************************************************************
 ***                    d i o d r v r _ i n t r                        ***
 *************************************************************************
 *
 *  Name:       diodrvr_intr
 *
 *  Purpose:    Our interrupt handler.
 *
 *  Returns:    None.
 *
 *************************************************************************/

void
diodrvr_intr( intr_arg_t arg )
{
        register card_t         *cp;
        register buf_t          *bp;
        register caddr_t        mite_adr_mite, dio_adr;

        register uint_t         intcsr, CHORbits, CHCRbits;
        register int            i, rw, err;
        register int            s;
        size_t                  p_size;
        alenaddr_t              p_addr;
        register int            count;
        register int            offset;
        size_t                          size;
        alenaddr_t          addr;
        alenlist_t                      unmappedList, mappedList;

        register uint_t         DMA_FLAGS = PCIIO_DMA_DATA | PCIIO_PREFETCH;

        unmappedList = alenlist_create( AL_NOCOMPACT );
        mappedList = alenlist_create( AL_NOCOMPACT );

        /*  get the lock  */
        cp = (card_t *)arg;
        nanotime ( & cp->intr_time);
        s = DIODRVR_LOCK(&cp->diodrvr_mlock);

        /*  a stray interrupt ?  */
        dio_adr = cp->dio_adr;
        intcsr = Inp8(dio_adr + (Group_1_Flags^3));
        if ( (intcsr & 0xE3) == 0 ) {
                DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);
                return;
        }

        /* report interrupt */
        #ifdef DEBUG
    printf ("diodrvrIntr: Count Expired\n");
        #endif

        bp = cp->bp;

        /*  cancel any outstanding timer  */
        printf("cancel timer\n");
        if ( cp->tid > 0 ) {
                untimeout(cp->tid);
                cp->tid = 0;
        }
```

```
        /* reset the MITE Interrupt */
        CHCRbits = (unsigned int) (DMA_CHCR_CLR_DONE_IE | DMA_CHCR_CLR_DMA_IE);
        Out32(DMA_BASE_ADDR + DMA_CHCR, CHCRbits);

        /* reset the DIO interrupt */
        Out8(cp->dio_adr + (Group_1_First_Clear^3), 0xF8); /* bits 4,5, 7 */
        Out8(cp->dio_adr + (Group_1_Second_Clear^3), 0x03);

        /* disarm the DMA and reset the MITE */
        CHORbits = (DMA_CHOR_DMARESET | DMA_CHOR_FRESET | DMA_CHOR_ABORT |
DMA_CHOR_CLR_SEND_TC);
        Out32(DMA_BASE_ADDR + DMA_CHOR, CHORbits);

        /* stop the DIO */
        Out8(dio_adr + (Protocol_Register_1_Group_1^3), 0x0);

         /*  single page DMA done. DMA the next page if any  */
        if (cp->dmatype == DMA_PAGE) {

                bp->b_resid -= cp->dmasize;
                cp->page_no--;

                if ( cp->page_no <= 0 ) {  /* no more pages */

                        #ifdef DEBUG
                                printf ("diodrvrIntr: No more pages to Dma\n");
                                printf ("diodrvrIntr: biodone() read/write\n");
                        #endif

                        alenlist_done( cp->addrList );
                        alenlist_done( cp->alenList );

                        cp->addrList = 0;
                        cp->alenList = 0;

                        cp->dmastat = DMA_IDLE;

                        /* free our DMA maps */
                        pciio_dmamap_free(cp->buffer_map);
                        pciio_dmamap_free(cp->link_map);

                        /* reset the DIO */
                        Out8(dio_adr + (Protocol_Register_1_Group_1^3), 0x0);

                        biodone(cp->bp);
                        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);
                        goto get_out;
                }

                /*  get next page to DMA  */
                #ifdef DEBUG
                        printf ("diodrvrIntr: DMA next page ..left = %d\n", cp->page_no-1 );
                #endif

                /* set up alenlist for next DMA page */
                count = 0;
                i = 0;
                for ( ;; ) {
                        if ( alenlist_get(cp->alenList, NULL, NBPP, &addr, &size, 0) !=
                                        ALENLIST_SUCCESS ) {
                                break;
                        }
                        if ((count += size) == cp->dma_page_size) {
                                /*offset = alenlist_cursor_offset( cp->addrList, NULL );
                                alenlist_cursor_init ( cp->addrList, offset -
sizeof(alenlist_t), NULL);*/
                                break;
                        }
                        printf("setting up DMA page in Intr i: %d  count: 0x%x  addr: 0x%x\n",
```

```
i, count, addr);

                        alenlist_append(unmappedList, addr, size, AL_NOCOMPACT);

                        i++;
                } /* for (;;) */

                alenlist_cursor_init ( unmappedList, NULL, NULL );

                pciio_dmamap_done( cp->buffer_map );

                mappedList = pciio_dmamap_list ( cp->buffer_map, unmappedList, DMA_FLAGS);

                if ( mappedList == (alenaddr_t)NULL ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create alenlist" );
                        bioerror ( bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(cp->buffer_map);
                        pciio_dmamap_free(cp->link_map);
                }

                alenlist_cursor_init ( mappedList, NULL, NULL );
                alenlist_get( mappedList, NULL, NBPP, &p_addr, &p_size, 0);
                p_size = count;

                printf("start single page DMA of size: 0x%x   addr: 0x%x\n" , p_size, p_addr);

                if ( (p_addr < 0x30000000) || (p_addr > 0x50000000) ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Incorrect DMA mapping. Exiting
driver!!");

                        alenlist_done( cp->alenList );
                        alenlist_done( mappedList );

                        pciio_dmamap_free( cp->buffer_map );
                        pciio_dmamap_free( cp->link_map );

                        bioerror ( bp, EIO);
                        biodone (bp);
                        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);

                        goto get_out;
                }

                if ( cp->dmastat == DMA_READ_WAIT )
                        rw = B_READ;
                else  rw = B_WRITE;

                diodrvr_int_countexp( cp );

                diodrvrStartSingleDma ( cp, p_addr, p_size, rw );

                if ( rw == B_READ )
                        cp->dmastat = DMA_READ_WAIT;
                else cp->dmastat = DMA_WRITE_WAIT;
                cp->chain_list    = NULL;

                /*  so we don't wait forever for an interrupt  */
                cp->tid = itimeout(diodrvrTimeOut, cp, RW_TIMER, pltimeout,  0, 0, 0);

                DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);

                goto get_out;

        }  /*** if ( cp->dmatype == DMA_PAGE ) ***/

        /* free up chain list structure */
        if (cp->dmatype == DMA_CHAIN)
                kmem_free ( cp->chain_list, cp->page_no * sizeof(diodrvr_dmapage_t) );
```

```
        alenlist_done( cp->addrList );
        alenlist_done( cp->alenList );

        cp->addrList = 0;
        cp->alenList = 0;

        cp->dmastat = DMA_IDLE;
        bp->b_resid -= cp->dmasize;

        /* free our DMA maps */
        pciio_dmamap_free(cp->buffer_map);
        pciio_dmamap_free(cp->link_map);

        /* reset the DIO */
        Out8(dio_adr + (Protocol_Register_1_Group_1^3), 0x0);

        biodone (cp->bp);
        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);

get_out:

        #ifdef DEBUG
    printf ( "diodrvrIntr exit\n" );
    #endif

}

/************************************************************************
 ***                  d i o d r v r _ u n l o a d                     ***
 ************************************************************************
 *
 *  Name:       diodrvr_unload
 *
 *  Purpose:    Unloads the driver.
 *
 *  Returns:    0 Success or errno
 *
 ************************************************************************/
int
diodrvr_unload(void)
{
        cmn_err (CE_NOTE, "Unloading the PCI_DIO_32HS Driver");
        return(0);
}

/************************************************************************
 ***                  d i o d r v r _ u n r e g                       ***
 ************************************************************************
 *
 *  Name:       diodrvr_unreg
 *
 *  Purpose:    Unregister the driver
 *
 *  Returns:    0 Success or errno
 *
 ************************************************************************/
int
diodrvr_unreg(void)
{
        cmn_err (CE_NOTE, "Unregisterning PCI-DIO-32HS Driver");
        pciio_driver_unregister(DRIVER_PREFIX);

        return(0);
}

/************************************************************************
 ***                  d i o d r v r _ o p e n                         ***
 ************************************************************************
```

```
 *
 *  Name:       diodrvr_open
 *
 *  Purpose:    Opens the card. This sample driver simply verifies that
 *              the card's info structure can be retrieved and checks
 *              the card's Base_Register.
 *
 *  Returns:    0 = Success, or errno.
 *
 ****************************************************************************/
int
diodrvr_open(dev_t *devp, int flag, int otyp, cred_t *cred)
{
        register card_t       *cp;
        register vertex_hdl_t vhdl;
        __userabi_t           uabi;

        /*   Get the vertex handle and pointer to card's info  */
        vhdl = dev_to_vhdl ( *devp );
        if (vhdl == NULL){
                cmn_err(CE_WARN, "diodrvr_open: dev_to_vhdl returns NULL" );
                return(EIO);
        }

        cp = (card_t *)device_info_get ( vhdl );


        /*  some error checking first    */
        if ( !(cp->status & CARD_ATTACHED) ) {
                cmn_err (CE_WARN, "diodrvr_open: Driver is not attached" );
                return (ENODEV);
        }

        if ( cp->status & CARD_OPEN) {
                cmn_err (CE_WARN, "diodrvr_open: Device is busy" );
                return (EBUSY);
        }

        /* diodrvrReset(cp);   reset the board  */
        cp->status |= CARD_OPEN;

        /*  default values */
        cp->dmatype = DMA_PAGE;
        /* cp->dmacmd  = DIODRVR_TRANSP; */

        #ifdef DEBUG
                printf("\n *** DIO card opened *** \n" );
        #endif

        /*  get user's ABI for correct struct usage in diodrvr_ioctl() */
        userabi(&uabi);
        if ( uabi.uabi_szptr == 8 )
                cp->user_abi = ABI_IRIX5_64;
        else    cp->user_abi = ABI_IRIX5_N32;

        /*  initialize our mutex lock and sv_t for sleep/wake  */
        DIODRVR_LOCK_INIT(&cp->diodrvr_mlock);
        SV_INIT(&cp->diodrvr_sv, SV_DEFAULT, "diodrvrsv" );


        /*  zero out all time measurements    */
        bzero ( &cp->start_time, sizeof(struct timespec) );
        bzero ( &cp->intr_time, sizeof(struct timespec) );
        bzero ( &cp->call_time, sizeof(struct timespec) );
        bzero ( &cp->ret_time, sizeof(struct timespec) );

        return(0);
}
```

```
/*****************************************************************
 ***                  d i o d r v r _ c l o s e                ***
 *****************************************************************
 *
 *  Name:       diodrvr_close
 *
 *  Purpose:    Closes the card. This sample driver's close statement
 *              prints out the addresses and Base_Register's value for
 *              verification.
 *
 *  Returns:    0 = Success, or errno
 *
 *****************************************************************/
int
diodrvr_close(dev_t dev, int flag, int otyp, cred_t *cred)
{
        register card_t       *cp;
        register vertex_hdl_t vhdl;

        /*  get the vertex handle and pointer to card's info  */
        vhdl = dev_to_vhdl ( dev );

        cp = (card_t  *)device_info_get ( vhdl );
        cp->status &= ~CARD_OPEN;

        /*  zero out all time measurements   */
        bzero ( &cp->start_time, sizeof(struct timespec) );
        bzero ( &cp->intr_time, sizeof(struct timespec) );
        bzero ( &cp->call_time, sizeof(struct timespec) );
        bzero ( &cp->ret_time, sizeof(struct timespec) );

        return(0);
}

/*****************************************************************
 ***                  d i o d r v r _ m a p                    ***
 *****************************************************************
 *
 *  Name:       diodrvr_map
 *
 *  Purpose:    Allocate a piece of continious memory and map it to user's
 *              address space.
 *
 *  Returns:    0 = Success, or errno
 *
 *****************************************************************/
int
diodrvr_map ( dev_t dev, vhandl_t *vh, off_t offset, size_t len, uint_t prot )
{
        register int            ret;
        register int            s;
        register caddr_t        kv;
        register vertex_hdl_t   vhdl;
        register card_t         *cp;

        /*  get the vertex handle and pointer to card's info  */
        vhdl = dev_to_vhdl ( dev );
        cp = (card_t  *)device_info_get ( vhdl );

        if ( cp->dio_adr == (caddr_t)NULL ) {
                cmn_err (CE_NOTE, "diodrvr_map: No DIO memory for mapping" );
                return (ENOMEM);
        }

        if ( len > (size_t)(DIO_RAM_SIZE) ) {
                cmn_err (CE_NOTE, "diodrvr_map: Only %d bytes available for map, requested %d
 bytes", DIO_RAM_SIZE, len );
                return (ENOMEM);
        }
```

```
        ret = v_mapphys( vh, (void *)cp->dio_adr, len );

        if ( ret > 0  ) {
                cmn_err (CE_WARN, "diodrvr_map: Could not map, ret = %d" , ret );
                return (ret);
        }

        /*  save for later */
        cp->vhandl = vh;

        cmn_err (CE_NOTE, "diodrvr_map: mapped %d bytes" , len );
        return (ret);
}

/*************************************************************************
 ***                  d i o d r v r _ u n m a p                        ***
 *************************************************************************
 *
 *  Name:       diodrvr_unmap
 *
 *  Purpose:    Unmap the kernel buffer we allocated before.
 *
 *  Returns:    Always 0. There is nothing to free up here.
 *
 *************************************************************************/
int
diodrvr_unmap ( dev_t dev, vhandl_t *vh )
{
        return (0);
}


/*************************************************************************
 ***                  d i o d r v r _ i o c t l                        ***
 *************************************************************************
 *
 *  Name:       diodrvr_ioctl
 *
 *  Purpose:    Handles user Ioctl command. These commands can be
 *              found in diodrvr_user.h.
 *
 *  Returns:    0 = Success, or errno
 *
 *************************************************************************/
int
diodrvr_ioctl(dev_t dev, int cmd, void *arg, int mode, cred_t *cred,
              int *rvalp )
{
        register card_t       *cp;
        register vertex_hdl_t vhdl;
        register uint_t      *tmp_ibuf;
        register int          tot_bytes, err, i, s;
        uint_t         tmp_int;

        /*  get the vertex handle and pointer to card's info  */
        vhdl = dev_to_vhdl ( dev );

        cp = (card_t  *)device_info_get ( vhdl );

        if ( cp == (card_t *)NULL ) {
                cmn_err (CE_WARN, "diodrvr_ioctl: Card_t is NULL");
                return (EIO);
        }

        /*  is device opened ? */
        if (!( cp->status & CARD_OPEN) ) {
                cmn_err (CE_NOTE, "diodrvr_ioctl: Device is not opened");
                return (EIO);
```

```
        }

        /* #ifdef DEBUG
        printf("ioctl arg: 0x%x\n", *((int *)arg));
        #endif */

        /* ====================
         *    Ioctl commands
         * ==================== */

        switch ( cmd ) {

                case DIODRVR_RESET:
                        diodrvr_reset ( cp );
                        break;

                case DIODRVR_BURST_CLKOUT:
                        diodrvr_burst_clkout( cp, arg );
                        break;

                case DIODRVR_BURST_CLKIN:
                        diodrvr_burst_clkin( cp, arg );
                        break;

                case DIODRVR_DMA_GROUP1:
                        diodrvr_dma_group1( cp, arg);
                        break;

                case DIODRVR_INTR_COUNTEXP:
                        diodrvr_int_countexp( cp );
                        break;

                case DIODRVR_RUN:
                        diodrvr_run( cp );
                        break;

                case DIODRVR_STOP:
                        diodrvr_stop( cp );
                        break;

                case DIODRVR_GETTIME:
                        diodrvrUserTime ( cp, arg );
                        break;

        }  /***  end switch   **/

        return(0);
}

/*************************************************************************
 ***                  d i o d r v r _ r e a d                          ***
 *************************************************************************
 *
 *  Name:       diodrvr_read
 *
 *  Purpose:    Read entry routine. DMAs data from the board to the
 *              user's address space.
 *
 *  Returns:    0 = Success, or errno
 *
 *************************************************************************/
int
diodrvr_read( dev_t dev, uio_t *uiop, cred_t *crp)
{
        register card_t  *cp;
        register vertex_hdl_t vhdl;
        register int          ret;

        /*   get to the board's info structure  */
```

```
        vhdl = dev_to_vhdl ( dev );

        cp = (card_t  *)device_info_get ( vhdl );
        cp->bp = (buf_t *)NULL;
        cp->addrList = 0;
        cp->dmastat = 0;
        cp->dmabits = 0;

        /*  do the transfer  */

        ret = uiophysio ( diodrvr_Strategy, NULL, dev, B_READ, uiop );


        ret = 0;

        if ( cp->addrList ) {
                alenlist_done(cp->addrList);
                cp->addrList = 0;
        }

        #ifdef DEBUGTIME
        diodrvrReportTime ( "diodrvrRead", cp, 1);
        #endif

        return ( ret );
}

/*************************************************************************
 ***                  d i o d r v r _ w r i t e                         ***
 *************************************************************************
 *
 *  Name:       diodrvr_write
 *
 *  Purpose:    Write entry routine. DMAs data from user's address space
 *              to the board.
 *
 *  Returns:    0 = Success, or errno
 *
 *************************************************************************/
int
diodrvr_write( dev_t dev, uio_t *uiop, cred_t *crp)
{
        register card_t       *cp;
        register vertex_hdl_t vhdl;
        register int          ret;

        /*   get to the board's info structure  */
        vhdl = dev_to_vhdl ( dev );
        cp = (card_t  *)device_info_get ( vhdl );
        cp->bp       = (buf_t *)NULL;
        cp->dmastat  = 0;
        cp->dmabits  = 0;
        cp->addrList = 0;

        cp->dmatype = DMA_PAGE;

        bzero ( &cp->start_time, sizeof(struct timespec) );
        bzero ( &cp->intr_time, sizeof(struct timespec) );
        bzero ( &cp->call_time, sizeof(struct timespec) );
        bzero ( &cp->ret_time, sizeof(struct timespec) );

        printf("entering diodrvrWrite routine\n");

        /*  do the transfer    */
        nanotime ( &cp->call_time );
        ret = uiophysio ( diodrvr_Strategy, NULL, dev, B_WRITE, uiop );
        nanotime ( &cp->ret_time );

        printf("diodrvr_Write: returned from diodrvrStrategy: %d\n" , ret);
```

```
        if ( cp->addrList ) {
                alenlist_done(cp->addrList);
                cp->addrList = 0;
        }

        #ifdef DEBUGTIME
                diodrvrReportTime ( "diodrvr_Write",  cp, 1);
        #endif

        printf("diodrvr_Write: returning from driver: %d\n", ret);
        return ( ret );

}


/*****************************************************************************
 ***                    d i o d r v r _ e r r o r                          ***
 *****************************************************************************
 *
 *  Name:       diodrvr_error
 *
 *  Purpose:    Traps PCI bus error.
 *
 *  Returns:    0 = Success, or errno
 *
 *****************************************************************************/

static int
diodrvr_error(void *einfo,
           int error_code,
           ioerror_mode_t mode,
           ioerror_t *ioerror)
{
        cmn_err (CE_NOTE, "diodrvr_error: called");
        printf ("error_code = %d (0x%x)\n", error_code, error_code );
        printf ("io_errortype = %d\n", ioerror->ie_errortype );
        printf ("io_busspace  = %d\n", ioerror->ie_busspace );
        printf ("io_busaddr   = 0x%x\n", ioerror->ie_busaddr );
        printf ("io_memaddr   = 0x%x\n", ioerror->ie_memaddr );
        printf ("error_mode   = %d (0x%x)\n", mode, mode );
        return(IOERROR_UNHANDLED);
}

/*
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~
   ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~    S u p p o r t i n g   R o u t i n e s      ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~
   ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

/*****************************************************************************
 ***                    d i o d r v r _ S t r a t e g y                    ***
 *****************************************************************************
 *
 *  Name:       diodrvr_Strategy
 *
 *  Purpose:    Strategy routine. It actually handles read/write
 *              and starts the DMA. It is called by uiophysio() kernel
 *              routine.
 *
 *  Returns:    0 = Success, or errno
 *
 *****************************************************************************/
static int
diodrvr_Strategy ( buf_t  *bp )
```

```
{
        register card_t                         *cp;
        register vertex_hdl_t           vhdl, conn;
        register diodrvr_dmapage_t  *dmaPg;
        register int                            i, ret, rw, tot_bytes;
        register int                            s;
        register uint_t                         fill_bits;
        alenlist_t                              addrList2;
        size_t                                  p_size;
        alenaddr_t                              p_addr;
        iopaddr_t                               p_dmaPg;

        pciio_dmamap_t                          buffer_map, link_map;
        device_desc_t                           dev_desc;

        register int                            count;
        register int                            offset;
        size_t                                          size;
        alenaddr_t                                      addr;

        register uint_t                         DMA_FLAGS = PCIIO_PREFETCH |
PCIIO_WRITE_GATHER;

        if ( !BP_ISMAPPED(bp) ) {
                cmn_err (CE_NOTE, "diodrvrStrategy: Unmapped buf_t used\n" );
                bioerror ( bp, EIO);
                biodone (bp);
                return(EIO);
        }

        printf("diodrvrStrategy: entering diodrvrStrategy\n" );

        /*   get to the board's info structure  */
        vhdl = dev_to_vhdl ( bp->b_edev );
        cp = (card_t  *)device_info_get ( vhdl );
        conn = cp->conn;

        cp->dma_page_size = 0x40000; /* 256 KB */

        printf("diodrvrStrategy: done getting board's info structure\n" );

        /*  clear error and save bp  */
        bioerror(bp, 0);
        bp->b_resid = bp->b_bcount;
        cp->bp = bp;

        if ( bp->b_flags & B_READ )
                rw = B_READ;    /*  board -> mem    */
        else    rw = B_WRITE;   /*  mem   -> board */

        printf("diodrvrStrategy: allocating DMA maps\n" );

        /* set up mapping device descriptor */
        dev_desc = 0;

        /* allocate our DMA maps */
        buffer_map = pciio_dmamap_alloc( conn, dev_desc, cp->dma_page_size + 0x6000, DMA_FLAGS
);
        if ( buffer_map == (pciio_dmamap_t)NULL ) {
                cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create buffer map" );
                bioerror ( bp, EIO);
                biodone (bp);
                return(EIO);
        }

        link_map = pciio_dmamap_alloc( conn, dev_desc, 0x1000, DMA_FLAGS );
        if ( link_map == (pciio_dmamap_t)NULL ) {
                cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create link map" );
                bioerror ( bp, EIO);
```

```
                biodone (bp);
                pciio_dmamap_free(buffer_map);
                return(EIO);
        }

        cp->buffer_map = buffer_map;
        cp->link_map = link_map;

        /*  create scatter-gather list   */
        cp->alenList = buf_to_alenlist ((alenlist_t)NULL, bp, AL_NOCOMPACT);
        addrList2 = alenlist_clone( cp->alenList, AL_NOCOMPACT);

        cp->addrList_page = alenlist_create( AL_NOCOMPACT );

        if ( (cp->dmatype == DMA_CHAIN) || (cp->dmatype == DMA_SINGLE) ) {
                #ifdef DEBUG
                        printf("diodrvrStrategy: translating DMA addresses for link
chaining\n");
                #endif
                cp->addrList = pciio_dmamap_list ( buffer_map, addrList2, DMA_FLAGS);
                if ( cp->addrList == (alenaddr_t)NULL ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create alenlist" );
                        bioerror ( bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
                }
        }

        printf("diodrvrStrategy: getting alenlist size\n" );

        #if 0
                cp->page_no  = alenlist_size ( cp->alenList );
        #endif

        if (cp->dmatype == DMA_PAGE) {
                cp->page_no = bp->b_bcount / cp->dma_page_size;
                if ((bp->b_bcount % cp->dma_page_size) !=0)
                        (cp->page_no)++;
        }
        else
                cp->page_no  = diodrvrAlenlistSize ( cp->alenList );

        printf("diodrvrStrategy: number of pages %d  DMAtype: %d\n" , cp->page_no, cp->dmatype
);

        /* =======================
         *    Chained DMA
         * ======================= */
        if ( cp->dmatype == DMA_CHAIN ) {

                printf("diodrvrStrategy: making chain\n" );

                dmaPg = diodrvrMakeChain ( cp, cp->addrList, cp->page_no );

                if ( dmaPg == (diodrvr_dmapage_t *)NULL ) {
                        cmn_err (CE_NOTE,
                                 "diodrvrStrategy: Error creating chain list" );
                        alenlist_done (cp->addrList);
                        bioerror (bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
                }

                tot_bytes = cp->page_no * sizeof(diodrvr_dmapage_t); /* length of link chain in
bytes */
```

```
                p_dmaPg = pciio_dmamap_addr ( link_map, kvtophys(dmaPg), tot_bytes);

                if ( p_dmaPg == (iopaddr_t)NULL) {
                        cmn_err (CE_NOTE,
                                 "diodrvrStrategy: Error creating chain list" );
                        alenlist_done (cp->addrList);
                        bioerror (bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
                }

                #if 0
                /*  write back cache for chain list  */
                dki_dcache_wbinval ( dmaPg, tot_bytes );
                #endif

                /*  start the Chained Dma  */
                s = DIODRVR_LOCK(&cp->diodrvr_mlock);

                diodrvrStartProgDma ( cp, p_dmaPg, bp->b_bcount, rw );

/*************************************
                This is the test section
*************************************

                cp->addrList = pciio_dmamap_list ( buffer_map, cp->alenList, PCIIO_DMA_DATA);
                if ( cp->addrList == (alenaddr_t)NULL ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create alenlist");
                        bioerror ( bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
                }

                alenlist_get( cp->addrList, NULL, NBPP, &addr, &size, 0);

                printf("addr: 0x%x   size: 0x%x\n", addr, size);

                if ( (addr < 0x20000000) || (addr > 0x60000000) ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Incorrect DMA mapping. Exiting
 driver!!");
                        bioerror ( bp, EIO);
                        biodone (bp);

                        alenlist_done( cp->addrList );
                        alenlist_done( cp->addrList_page );

                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
                }

                diodrvrStartSingleDma ( cp, addr, bp->b_bcount, rw );

*************************************
                End of test section
*************************************/

                if ( rw == B_READ )
                     cp->dmastat = DMA_READ_WAIT;
                else cp->dmastat = DMA_WRITE_WAIT;
                cp->chain_list = (caddr_t)dmaPg;

                printf("diodrvrStrategy: wait for timeout\n" );

                /*  so we dont wait forever for an interrupt  */
```

```
        cp->tid = itimeout(diodrvrTimeOut, cp, RW_TIMER, pltimeout,  0, 0, 0);

        printf("diodrvrStrategy: unlock memory\n");
        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);

        printf("diodrvrStrategy: return to caller\n");
        return(0);
    }    /* end of chained DMA */


    /* ============================
     *      Single page DMA
     * ============================ */

    /*  get a page to DMA  */
    if (cp->dmatype == DMA_PAGE) {
        printf("diodrvrStrategy: setting up page to DMA\n");
        count = 0;
        i = 0;
        alenlist_cursor_init ( cp->alenList, NULL, NULL );
        for ( ;; ) {
            if ( alenlist_get( cp->alenList, NULL, NBPP, &addr, &size, 0) !=
                    ALENLIST_SUCCESS ) {
                break;
            }

            count += size;
            alenlist_append(cp->addrList_page, addr, size, AL_NOCOMPACT);
            i++;
            printf("setting up DMA page i: %d  count: 0x%x  addr: 0x%x\n" , i,
count, addr);

            if (count > cp->dma_page_size) {
                printf("alenlist create reset\n");
                break;
            }
        } /* for (;;) */

        alenlist_cursor_init ( cp->addrList_page, NULL, NULL );

        cp->addrList = pciio_dmamap_list ( buffer_map, cp->addrList_page, DMA_FLAGS);

        if ( cp->addrList == (alenaddr_t)NULL ) {
            cmn_err (CE_NOTE, "diodrvrStrategy: Cannot create map alenlist" );
            bioerror ( bp, EIO);
            biodone (bp);
            pciio_dmamap_free(buffer_map);
            pciio_dmamap_free(link_map);
            return(EIO);
        }

        alenlist_get(cp->addrList, NULL, NBPP, &p_addr, &p_size, 0);
        p_size = count;

    } /* end if (cp->dmatype == DMA_PAGE) */

    else {                  /* only gets here is dmatype == DMA_SINGLE */
        if ( alenlist_get(cp->addrList, NULL, NBPP,
                &p_addr, &p_size, 0) != ALENLIST_SUCCESS ) {
            cmn_err (CE_WARN, "diodrvrDma: Not enough Memory" );
            alenlist_done(cp->addrList);
            bioerror ( bp, EIO);
            biodone(bp);
            return(EIO);
        }
    }

    /*  single dma, memory -> board  */
    printf("start single page DMA of size: 0x%x   addr: 0x%x\n" , p_size, p_addr);
```

```
        if ( (p_addr < 0x20000000) || (p_addr > 0x60000000) ) {
                        cmn_err (CE_NOTE, "diodrvrStrategy: Incorrect DMA mapping. Exiting
 driver!!");
                        bioerror ( bp, EIO);
                        biodone (bp);
                        pciio_dmamap_free(buffer_map);
                        pciio_dmamap_free(link_map);
                        return(EIO);
        }

        s = DIODRVR_LOCK(&cp->diodrvr_mlock);
        diodrvrStartSingleDma ( cp, p_addr, p_size, rw );

        if ( rw == B_READ )
                cp->dmastat = DMA_READ_WAIT;
        else cp->dmastat = DMA_WRITE_WAIT;

        cp->chain_list    = NULL;

        /*  so we don't wait forever for an interrupt  */
        cp->tid = itimeout(diodrvrTimeOut, cp, RW_TIMER, pltimeout,  0, 0, 0);

        printf("diodrvrStrategy: return from caller\n" );
        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);

        return(0);
}

/*************************************************************************
 ***                    d i o d r v r M a k e C h a i n                 ***
 *************************************************************************
 *
 *  Name:       diodrvrMakeChain
 *
 *  Purpose:    given an alenlist, creates chained list for Prog DMA.
 *
 *  Returns:    NULL for error or address of chained list
 *
 *************************************************************************/

static diodrvr_dmapage_t *
diodrvrMakeChain ( card_t *cp, alenlist_t addrList, int page_no )
{
        register                diodrvr_dmapage_t * dmaPg;
        register int    i, tot_words;
        size_t                  p_size;
        alenaddr_t              p_addr;
        iopaddr_t               pa;

        printf("dioMakeChain: entering dioMakeChain\n" );

        dmaPg = (diodrvr_dmapage_t *)kmem_alloc ((page_no+1) * sizeof(diodrvr_dmapage_t),
                                KM_NOSLEEP | KM_PHYSCONTIG | KM_CACHEALIGN);

        if ( dmaPg == (diodrvr_dmapage_t *)NULL ) {
                cmn_err (CE_NOTE, "diodrvrMakeChain: Not enough mem for chained list" );
                return( (diodrvr_dmapage_t *)NULL);
        }

        /*  fill the chained list with address-size values */
        for ( i = 0; i < page_no; i++ ) {
                if ( alenlist_get(addrList, NULL, NBPP, &p_addr, &p_size, 0) !=
 ALENLIST_SUCCESS ) {
                        cmn_err (CE_WARN, "diodrvrMakeChain: Bad alenlist\n" );
                        return( (diodrvr_dmapage_t *)NULL);
                }

                pa  = pciio_dmatrans_addr ( cp->conn, 0, kvtophys(&dmaPg[i+1]),
```

```
sizeof(diodrvr_dmapage_t), PCIIO_DMA_DATA);
                tot_words          = (int)p_size/sizeof(uint_t);

                dmaPg[i].size      = p_size; /*tot_words;*/
                dmaPg[i].addr      = p_addr;
                dmaPg[i].DAR       = 0x1C;
                dmaPg[i].nextaddr = pa;

                #ifdef DEBUG
                printf("diodrvrMakeChain: page no: %d  size: 0x%x   addr: 0x%x   next addr:
 0x%x\n", i, dmaPg[i].size, dmaPg[i].addr, dmaPg[i].nextaddr);
                #endif

        }
        dmaPg[page_no].size     = 0;
        dmaPg[page_no].nextaddr = 0;
        dmaPg[page_no].DAR              = 0;
        dmaPg[page_no].addr             = 0;

        return ( dmaPg );
}

/*************************************************************************
 ***             d i o d r v r S t a r t P r o g D m a                  ***
 *************************************************************************
 *
 *  Name:       diodrvrStartProgDma
 *
 *  Purpose:    Programs the board for Chained DMA and starts the Dma
 *
 * Returns:     None.
 *
 ************************************************************************/

static void
diodrvrStartProgDma ( card_t *cp, iopaddr_t p_dmaPg, int tot_bytes, int rw )
{
        register caddr_t        dio_adr, mite_adr;
        register uint_t         adr_bits, enable_bits, dmacfg, dmacmd, dmabits;

        register uint_t         CHORbits, CHCRbits, MCRbits, DCRbits, LKCRbits, LKARbits,
 LCR2bits;

        #ifdef DEBUG
                printf("diodrvrStartProgDma: entering diodrvrStartProgDma\n" );
                printf("diodrvrStartProgDma: number of bytes is 0x%x\n" , tot_bytes);
        #endif

        dio_adr  = cp->dio_adr;
        mite_adr = cp->mite_adr;
        dmacfg   = cp->dmacfg;
        dmacmd   = cp->dmacmd;
        dmabits  = cp->dmabits;
        cp->dmasize = tot_bytes; /* number of bytes to transfer by DMA */

        printf("diodrvrStarProgDma: MITE Dma addr: 0x%x\n" , DMA_BASE_ADDR);

        /* ======================
         *    memory -> board
         * ====================== */

        /* Build the CHOR bit pattern. */
        CHORbits = (unsigned int) (DMA_CHOR_DMARESET | DMA_CHOR_FRESET | DMA_CHOR_SET_SEND_TC);

        /* Build the CHCR bit pattern. */
        CHCRbits = (unsigned int) (DMA_CHCR_LINKLONG | DMA_CHCR_BURSTEN | DMA_CHCR_SET_DONE_IE
 | DMA_CHCR_SET_DMA_IE     );

        /* if (direction == INPUT)
```

```
        CHCRbits |= DMA_CHCR_DIR; */

    /* Build the MCR bit pattern. */
    MCRbits = DMA_MCR_RL64 | DMA_MCR_ASEQxP1 | DMA_MCR_PSIZEWORD | DMA_MCR_BLOCKEN;

    /* Build the DCR bit pattern. */
    DCRbits = DMA_DCR_RL64 | DMA_DCR_PSIZEWORD | DMA_DCR_PORTIO | MITE_DMA_AMDEVICE;

    /* Set the drq request line to use */
    switch (drqnum)
    {
            case 0:
                    DCRbits |= DMA_DCR_REQSDRQ0;
                    break;
            case 1:
                    DCRbits |= DMA_DCR_REQSDRQ1;
                    break;
            case 2:
                    DCRbits |= DMA_DCR_REQSDRQ2;
                    break;
            case 3:
                    DCRbits |= DMA_DCR_REQSDRQ3;
                    break;
    };

    /* Build the LKCR bit pattern. */
    LKCRbits = (uint_t) DMA_LKCR_RL64 | DMA_LKCR_ASEQUP | DMA_LKCR_PSIZEWORD;

    /* Build the LKAR bit pattern (start address of chained list) */
    LKARbits = (uint_t) p_dmaPg;

    #ifdef DEBUG
            printf("diodrvrStartProgDma: DMA link chain phys addr 0x%x\n" , LKARbits);
    #endif

    /* Read_Important_Registers(cp); */

    printf("writing DMA registers\n");


    /* Write the bit patterns to the registers. */
    Out32(DMA_BASE_ADDR + DMA_CHOR, CHORbits);
    Out32(DMA_BASE_ADDR + DMA_CHCR, CHCRbits);
    Out32(DMA_BASE_ADDR + DMA_MCR,  MCRbits);
    Out32(DMA_BASE_ADDR + DMA_DCR,  DCRbits);
    Out32(DMA_BASE_ADDR + DMA_LKCR, LKCRbits);
    Out32(DMA_BASE_ADDR + DMA_LKAR, LKARbits);

    /* Read_Important_Registers(cp); */

    /*  set Transfer count  */
    Out32(dio_adr + Group_1_Transfer_Count, tot_bytes);

    printf("arming the transfer\n");

    /* arm the DMA transfer */
    CHORbits = DMA_CHOR_START;
    Out32(DMA_BASE_ADDR + DMA_CHOR, CHORbits);

    Read_Important_Registers(cp);

    printf("starting the transfer\n");

    /* start the DMA transfer */
    Out8(dio_adr + (Protocol_Register_1_Group_1^3), 0x0F);

    /* Read_Important_Registers(cp);*/
}
```

```
/*****************************************************************************
 ***              d i o d r v r S t a r t S i n g l e D m a                 ***
 *****************************************************************************
 *
 *  Name:       diodrvrStartSingleDma
 *
 *  Purpose:    Programs the board for Single page DMA (read or write)
 *
 * Returns:     None.
 *
 *****************************************************************************/
static void
diodrvrStartSingleDma ( card_t *cp, alenaddr_t p_addr, size_t p_size, int rw )
{

        register caddr_t  dio_adr, mite_adr;
        register uint_t   adr_bits, enable_bits, dmacfg, dmacmd, dmabits, temp;
        register int      tot_words;

        register uint_t         CHORbits, CHCRbits, MCRbits, DCRbits, LKCRbits, LKARbits,
 MARbits, TCRbits;

        dio_adr  = cp->dio_adr;
        mite_adr = cp->mite_adr;
        dmacfg   = cp->dmacfg;
        dmacmd   = cp->dmacmd;
        dmabits  = cp->dmabits;
        enable_bits = 0;
        cp->dmasize = p_size;

        printf("diodrvrStartSingleDma: starting single DMA\n" );

        tot_words = (int)p_size/sizeof(uint_t);

        /* ======================
         *    memory -> board
         * ====================== */

        /* Build the CHOR bit pattern. */
        CHORbits = DMA_CHOR_DMARESET | DMA_CHOR_FRESET | DMA_CHOR_SET_SEND_TC;

        /* Build the CHCR bit pattern. */
        CHCRbits = DMA_CHCR_SET_DONE_IE | DMA_CHCR_SET_DMA_IE | DMA_CHCR_BURSTEN ;

        /* if (direction == INPUT)
                CHCRbits |= DMA_CHCR_DIR; */

        /* Build the MCR bit pattern. */
        MCRbits = DMA_MCR_RL64 | DMA_MCR_ASEQxP1 | DMA_MCR_PSIZEWORD | DMA_MCR_BLOCKEN;

        /* Build the DCR bit pattern. */
        DCRbits = DMA_DCR_RL64 | DMA_DCR_PORTIO | DMA_DCR_PSIZEWORD | DMA_DCR_AMDEVICE |
 DMA_DCR_BLOCKEN;

        /* Set the drq request line to use   */
        switch (drqnum)
        {
                case 0:
                        DCRbits |= DMA_DCR_REQSDRQ0;
                        break;
                case 1:
                        DCRbits |= DMA_DCR_REQSDRQ1;
                        break;
                case 2:
                        DCRbits |= DMA_DCR_REQSDRQ2;
                        break;
                case 3:
                        DCRbits |= DMA_DCR_REQSDRQ3;
```

```
                break;
        };


        /* DCRbits |= DMA_DCR_REQSINTMAX; */

        /* Build the LKCR bit pattern. */
        LKCRbits = (uint_t) DMA_LKCR_RL64 | DMA_LKCR_ASEQUP | DMA_LKCR_PSIZEWORD;

        /* Build the MAR, TCR amd LKAR bit patterns */
        MARbits = (uint_t) p_addr;
        TCRbits = (uint_t) p_size;
        LKARbits = (uint_t) 0;

        #ifdef DEBUG
                printf("diodrvrStartSingleDma: DMA buffer phys addr 0x%x  size: 0x%x\n" ,
MARbits, TCRbits);
        #endif

        printf("writing DMA registers\n");


        /* Write the bit patterns to the MITE DMA registers. */
        Out32(DMA_BASE_ADDR + DMA_CHOR, CHORbits);
        Out32(DMA_BASE_ADDR + DMA_CHCR, CHCRbits);

        Out32(DMA_BASE_ADDR + DMA_MCR,  MCRbits);

        Out32(DMA_BASE_ADDR + DMA_DCR,  DCRbits);
        Out32(DMA_BASE_ADDR + DMA_LKCR, LKCRbits);


        Out32(DMA_BASE_ADDR + DMA_MAR, MARbits);
        Out32(DMA_BASE_ADDR + DMA_TCR, TCRbits);
        Out32(DMA_BASE_ADDR + DMA_LKAR, LKARbits);

        printf("arming the transfer\n");

        /* arm the DMA transfer */
        CHORbits = DMA_CHOR_START;
        Out32(DMA_BASE_ADDR + DMA_CHOR, CHORbits);

        /*  set Transfer count  */
        Out32(dio_adr + Group_1_Transfer_Count, p_size);

        printf("starting the transfer\n");\

        Read_Important_Registers(cp);

        /* start the DMA transfer */
        Out8(dio_adr + (Protocol_Register_1_Group_1^3), 0x0F);
}

/*************************************************************************
 ***            d i o d r v r A l e n l i s t S i z e                  ***
 *************************************************************************
 *
 *  Name:       cocoAlenlistSize
 *
 *  Purpose:    Returns number of pairs in a given alenlist.
 *
 *  Returns:    Number of address/size entries
 *
 *************************************************************************/
static int
diodrvrAlenlistSize ( alenlist_t  al )
{
        register int        count;
        size_t              size;
```

```
        alenaddr_t          addr;

        printf("diodrvrAlenlistSize\n");

        alenlist_cursor_init ( al, NULL, NULL );
        count = 0;
        for ( ;; ) {
                if ( alenlist_get(al, NULL, NBPP, &addr, &size, 0) !=
                                ALENLIST_SUCCESS ) {
                        break;
                }
                count++;
        }
        alenlist_cursor_init ( al, NULL, NULL );

        return (count);
}

/************************************************************************
 ***                  d i o d r v r T i m e O u t                      ***
 ************************************************************************
 *
 *  Name:       cocoTimeOut
 *
 *  Purpose:    We timedout waiting for a read/write interrupt.
 *
 *  Returns:    None.
 *
 ************************************************************************/
static void
diodrvrTimeOut ( card_t *cp )
{
        register uint_t  intcsr;
        register int     s;

        /*  someone has the lock ? ignore the timeout  */
        if ( ( s = DIODRVR_TRYLOCK(&cp->diodrvr_mlock)) == 0 )
                return;

        cmn_err (CE_NOTE, "diodrvrTimeOut: Read/Write Timed out" );

        alenlist_done ( cp->addrList );
        cp->addrList = 0;

        alenlist_done ( cp->alenList );
        cp->alenList = 0;

        /* free our DMA maps */
        pciio_dmamap_free(cp->buffer_map);
        pciio_dmamap_free(cp->link_map);

        cp->iostat   = IO_TIME;

        bioerror ( cp->bp, ETIME);
        biodone (cp->bp);
        DIODRVR_UNLOCK(&cp->diodrvr_mlock, s);
}


/************************************************************************
***                  d i o d r v r_ r e s e t
***
************************************************************************/
void diodrvr_reset(card_t *cp)
{
        #ifdef DEBUG
                printf("diodrvr_reset command issued.\n" );
        #endif
```

```
        /* stop all transfers */
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x0);
        Out8(cp->dio_adr + (Protocol_Register_1_Group_2^3), 0x0);

        /* Flush FIFO Buffer and clear flags */
        Out8(cp->dio_adr + (Group_1_First_Clear^3), 0xFF); /* bits 4,5, 7 */
        Out8(cp->dio_adr + (Group_2_First_Clear^3), 0xFF);

        Out8(cp->dio_adr + (Group_1_Second_Clear^3), 0x03);
        Out8(cp->dio_adr + (Group_2_Second_Clear^3), 0x03);

        /* clear groups */
        Out8(cp->dio_adr + (Data_Path_Group_1^3),  0x00);
        Out8(cp->dio_adr + (Data_Path_Group_2^3),  0x00);

        /* write zero to the output ports */
        Out32(cp->dio_adr + Port_A_Output,  0x00);

}

/*************************************************************************
***        d i o d r v r_ b u r s t _ c l k o u t
***
*************************************************************************/
void diodrvr_burst_clkout(card_t *cp, int *arg)
{
        char width;

        #ifdef DEBUG
                printf("Executing diodrvr_burst_protocol_setup\n");
        #endif

        /* Configure the pins on ports ABCD simultaneously */
        Out32(cp->dio_adr + Port_A_Pin_Directions, 0x00);
        Out32(cp->dio_adr + Port_A_Pin_Mask, 0x00);
        Out32(cp->dio_adr + Port_A_Pin_Directions, 0xFFFFFFFF);

        /*** Reset  ***/
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x00);

        /*** Configure data paths and ports ***/
        Out8(cp->dio_adr + (Data_Path_Group_1^3), *arg);

        width =  (((*arg) & 0x8)>>3) + (((*arg) & 0x4)>>2) + (((*arg) & 0x2)>>1) + ((*arg) &
0x1);
        switch (width)
        {
                case 1:
                        width = 2; /* 8-bit transfers */
                        break;
                case 2:
                        width = 3; /* 16-bit transfers */
                        break;
                case 3:
                case 4:
                        width = 0; /* 32-bit transfers */
                        break;

        }

        width = (*arg & 0x80) >> 2 | width; /* this sets bit 5 if transfer is output */

        #ifdef DEBUG
                printf("protocol group: 0x%x\n", *arg);  /* should read 0x8f for 32-bit output
*/
                printf("protocol width: 0x%x\n", width); /* should read 0x20 for 32-bit output
*/
        #endif
```

```
        /*** Configure transfer width and RequireRLevel ***/
        Out8(cp->dio_adr + (Transfer_Size_Control_Group_1^3), width);

        /*** Set protocol to burst mode ***/
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_2_Group_1^3), 0x60);
        Out8(cp->dio_adr + (Protocol_Register_3_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_4_Group_1^3), 0x08);
        Out8(cp->dio_adr + (Protocol_Register_5_Group_1^3), 0x04);
        Out8(cp->dio_adr + (Protocol_Register_6_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_7_Group_1^3), 0x60);
        Out8(cp->dio_adr + (Protocol_Register_8_Group_1^3), 0x01);

        Out8(cp->dio_adr + (Clock_Speed_Group_1^3), 0x00);
        Out8(cp->dio_adr + (FIFO_Control_Group_1^3), 0x01);

}

/*************************************************************************
***        d i o d r v r_ b u r s t _ c l k i n
***
*************************************************************************/
void diodrvr_burst_clkin(card_t *cp, int *arg)
{
        char width;

        #ifdef DEBUG
                printf("Executing diodrvr_burst_protocol_setup\n");
        #endif

        /* Configure the pins on ports ABCD simultaneously */
        Out32(cp->dio_adr + Port_A_Pin_Directions, 0x00);
        Out32(cp->dio_adr + Port_A_Pin_Mask, 0x00);
        Out32(cp->dio_adr + Port_A_Pin_Directions, 0xFFFFFFFF);

        /*** Reset  ***/
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x00);

        /*** Configure data paths and ports ***/
        Out8(cp->dio_adr + (Data_Path_Group_1^3), *arg);

        width =  (((*arg) & 0x8)>>3) + (((*arg) & 0x4)>>2) + (((*arg) & 0x2)>>1) + ((*arg) &
 0x1);
        switch (width)
        {
                case 1:
                        width = 2; /* 8-bit transfers */
                        break;
                case 2:
                        width = 3; /* 16-bit transfers */
                        break;
                case 3:
                case 4:
                        width = 0; /* 32-bit transfers */
                        break;

        }

        width = (*arg & 0x80) >> 2 | width; /* this sets bit 5 if transfer is output */

        #ifdef DEBUG
                printf("protocol group: 0x%x\n", *arg);  /* should read 0x8f for 32-bit output
 */
                printf("protocol width: 0x%x\n", width); /* should read 0x20 for 32-bit output
 */
        #endif

        /*** Configure transfer width and RequireRLevel ***/
        Out8(cp->dio_adr + (Transfer_Size_Control_Group_1^3), width);
```

```
        /*** Set protocol to burst mode ***/
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_2_Group_1^3), 0x10);
        Out8(cp->dio_adr + (Protocol_Register_3_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_4_Group_1^3), 0x20);
        Out8(cp->dio_adr + (Protocol_Register_5_Group_1^3), 0x04);
        Out8(cp->dio_adr + (Protocol_Register_6_Group_1^3), 0x00);
        Out8(cp->dio_adr + (Protocol_Register_7_Group_1^3), 0x60);
        Out8(cp->dio_adr + (Protocol_Register_8_Group_1^3), 0x01);

        Out8(cp->dio_adr + (Clock_Speed_Group_1^3), 0x00);
        Out8(cp->dio_adr + (FIFO_Control_Group_1^3), 0x01);

}

/*************************************************************************
***             d i o d r v r_ d m a _ g r o u p 1
***
*************************************************************************/
void diodrvr_dma_group1(card_t *cp, int *arg)
{
        #ifdef DEBUG
                printf("progamming group1 Dma\n");
        #endif

        /* Specify DMA Channel to use (5=both channels) */
        Out8(cp->dio_adr + (DMA_Line_Control_Group_1^3), *arg);
}


/*************************************************************************
***        d i o d r v r_ i n t r _ c o u n t e x p
***
*************************************************************************/
void diodrvr_int_countexp(card_t *cp )
{
        /* Program the interrupt to occur on CountExpired */
        Out8(cp->dio_adr + (Interrupt_Enable_Group_1^3), 0x02); /* this enables bit 1 */

        /* Enable the interrupt line for normal operation */
        Out8(cp->dio_adr + (Interrupt_Control^3), 0x08); /* this enables bit 3 */
}


/*************************************************************************
***                d i o d r v r_ r u n
***
*************************************************************************/
void diodrvr_run(card_t *cp )
{
        /* start the transfer in numbered mode */
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x0F);
}

/*************************************************************************
***                d i o d r v r_ s t o p
***
*************************************************************************/
void diodrvr_stop(card_t *cp )
{
        /* start the transfer in numbered mode */
        Out8(cp->dio_adr + (Protocol_Register_1_Group_1^3), 0x0);
}

/*************************************************************************
 *               Read_Important_Registers
 *
 *************************************************************************/
```

```
void Read_Important_Registers(card_t *cp)
{
        uint_t MARcontents, TCRcontents, LKARcontents;
        uint_t  CHOR, CHCR, MCR, DCR, LKCR;

        #ifdef DEBUG
                printf("Reading important addresses\n");


        CHOR = Inp32(DMA_BASE_ADDR + DMA_CHOR);
        CHCR = Inp32(DMA_BASE_ADDR + DMA_CHCR);
        MCR = Inp32(DMA_BASE_ADDR + DMA_MCR);
        DCR = Inp32(DMA_BASE_ADDR + DMA_DCR);
        LKCR = Inp32(DMA_BASE_ADDR + DMA_LKCR);
        TCRcontents = Inp32(DMA_BASE_ADDR + DMA_TCR);
        MARcontents = Inp32(DMA_BASE_ADDR + DMA_MAR);
        LKARcontents = Inp32(DMA_BASE_ADDR + DMA_LKAR);
/*
        printf("CHOR contents =  0x%x\n", CHOR);
        printf("CHCR contents =  0x%x\n", CHCR);
        printf("MCR contents =  0x%x\n", MCR);
        printf("DCR contents =  0x%x\n", DCR);
        printf("LKCR contents =  0x%x\n", LKCR);
*/
        printf("TCR contents =  0x%x\n", TCRcontents);
        printf("MAR contents =  0x%x\n", MARcontents);
        printf("LKAR contents = 0x%x\n", LKARcontents);
        #endif

}

#ifdef DEBUGTIME
/*************************************************************************
 ***                  d i o d r v r R e p o r t T i m e                 ***
 *************************************************************************
 *
 *  Name:       diodrvrReportTime
 *
 *  Purpose:    Displays start, intr and end time of Dma
 *
 *  Returns:    None.
 *
 ************************************************************************/
static void
diodrvrReportTime ( caddr_t title, card_t *cp, int final )
{

        register struct timespec  *tv;
        struct timespec  dtv;
        diodrvr_timeval_t  st;
        diodrvr_timeval_t  et;
        diodrvr_timeval_t  dt;

        /*  report start and interrupt of DMA   */
        if ( final == 0 ) {
             diodrvrDiffTime ( &cp->start_time, &cp->intr_time, &dtv );
             diodrvrConvTime ( &dtv, &dt );
             diodrvrConvTime ( &cp->start_time, &st );
             diodrvrConvTime ( &cp->intr_time, &et );

             printf ("\n%s: %s DMA, %d Bytes\n",
             title, cp->dmatype == DMA_CHAIN ? "Chain":"Single", cp->dmasize );

             printf ("Start:\t%8dmsec %8dusec %8dnsec\n",
                st.diodrvr_msec, st.diodrvr_usec, st.diodrvr_nsec);
             printf ("Intrp:\t%8dmsec %8dusec %8dnsec\n",
                et.diodrvr_msec, et.diodrvr_usec, et.diodrvr_nsec);
             printf ("Diff:\t%8dmsec %8dusec %8dnsec\n",
                dt.diodrvr_msec, dt.diodrvr_usec, dt.diodrvr_nsec );
```

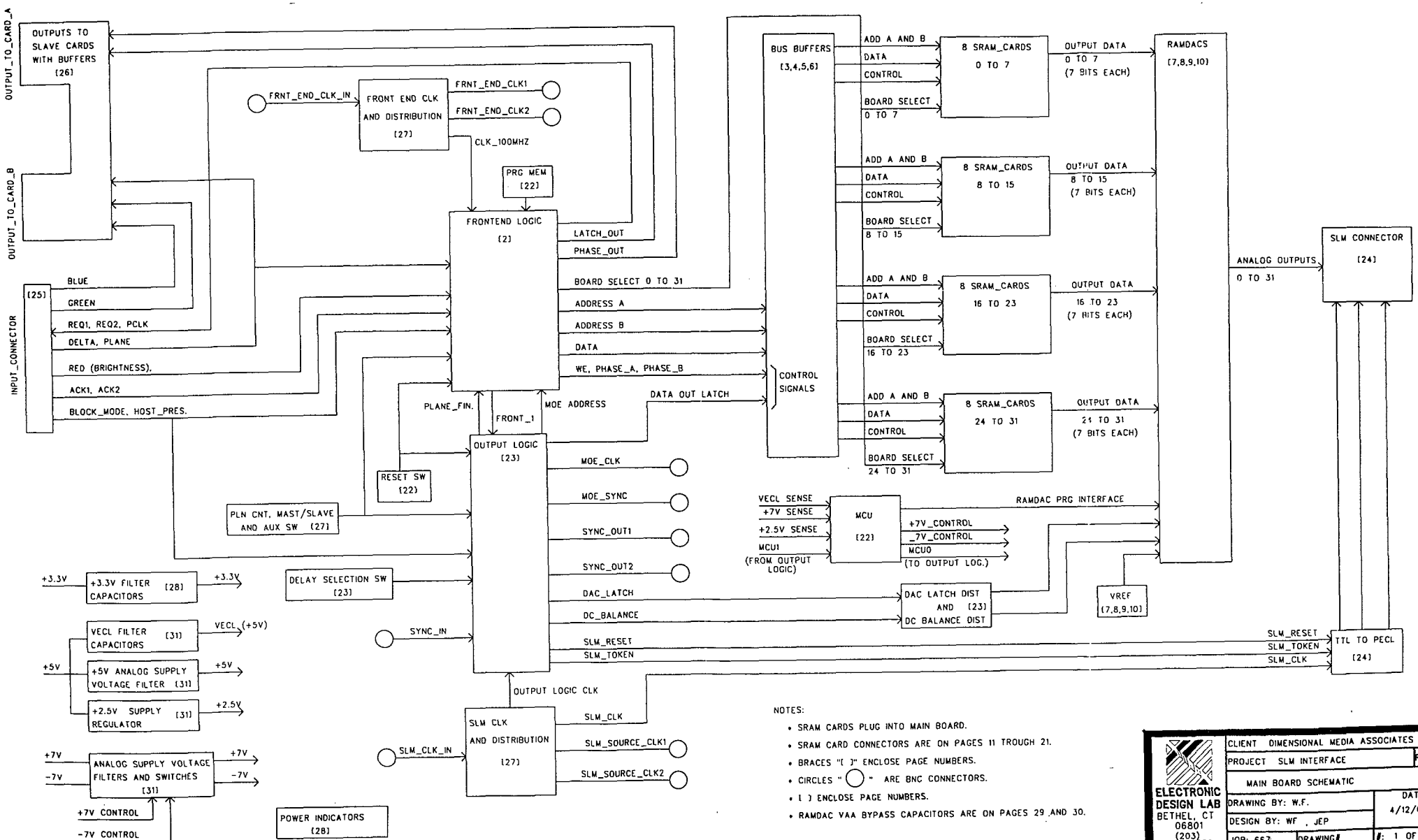
```
        return;
    }


    /*  report the time difference of call came in and return time */
    diodrvrDiffTime ( &cp->call_time, &cp->ret_time, &dtv );
    diodrvrConvTime ( &dtv, &dt );
    diodrvrConvTime ( &cp->call_time, &st );
    diodrvrConvTime ( &cp->ret_time, &et );

    printf ("\n%s:  final Call analysis\n", title );
    printf ("Call:\t%8dmsec %8dusec %8dnsec\n",
            st.diodrvr_msec, st.diodrvr_usec, st.diodrvr_nsec );
    printf ("Ret:\t%8dmsec %8dusec %8dnsec\n",
            et.diodrvr_msec, et.diodrvr_usec, et.diodrvr_nsec );
    printf ("Diff:\t%8dmsec %8dusec %8dnsec\n",
            dt.diodrvr_msec, dt.diodrvr_usec, dt.diodrvr_nsec );

}
#endif

/*************************************************************************
 ***                  d i o d r v r U s e r T i m e                     ***
 *************************************************************************
 *
 *  Name:       diodrvrUserTime
 *
 *  Purpose:    Returns timing to user at end.
 *
 *  Returns:    None.
 *
 *************************************************************************/
static void
diodrvrUserTime ( card_t *cp, int *arg )
{

    struct timespec  dtv;
    diodrvr_timeval_t  dt;

    /*  report the time difference of call came in and return time */
    diodrvrDiffTime ( &cp->call_time, &cp->ret_time, &dtv );
    diodrvrConvTime ( &dtv, &dt );

    *arg = dt.diodrvr_msec* 1000000 + dt.diodrvr_usec * 1000 + dt.diodrvr_nsec;

}


/*************************************************************************
 ***                  d i o d r v r D i f f T i m e                     ***
 *************************************************************************
 *
 *  Name:       diodrvrDiffTime
 *
 *  Purpose:    Given two timespec struct, it calculates the difference
 *
 *  Returns:    None.
 *
 *************************************************************************/

static void
diodrvrDiffTime ( struct timespec *st, struct timespec *et, struct timespec *dt )
{

    register long   s_sec, e_sec;
    register long   s_totn, e_totn, diff_n;

    s_sec    = st->tv_sec;
    s_totn   = (s_sec * 1000000000L) + st->tv_nsec;
```

```
        e_sec    = et->tv_sec;
        e_totn   = (e_sec * 1000000000L) + et->tv_nsec;

        diff_n   = e_totn - s_totn;

        dt->tv_sec = 0;
        dt->tv_nsec = diff_n;

        if ( diff_n > 1000000000L ) {
             dt->tv_sec  = diff_n  / 1000000000L;
             dt->tv_nsec = diff_n  % 1000000000L;
        }
}


/*************************************************************************
 ***                  d i o d r v r C o n v T i m e                    ***
 *************************************************************************
 *
 *  Name:       diodrvrConvTime
 *
 *  Purpose:    Converts a timespec_t to diodrvr_timeval_t
 *
 *  Returns:    None.
 *
 *************************************************************************/
static void
diodrvrConvTime ( struct timespec *ts, diodrvr_timeval_t *ct )
{
        long   totn, sec, msec, usec, nsec;

        totn = (ts->tv_sec * 1000000000L) + ts->tv_nsec;
        sec = msec = usec = nsec = 0;

        /*  round up nano to micro */
        if ( totn > 1000L ) {
                usec = totn / 1000L;
                nsec = totn % 1000L;
        }

        /*  round up micro to mili  */
        if ( usec > 1000L ) {
                msec = usec / 1000L;
                usec = usec % 1000L;
        }


        /*  round up mili to seconds  */
        if ( msec > 1000L ) {
                sec  = msec / 1000L;
                msec = msec % 1000L;
        }

        ct->diodrvr_sec  = sec;
        ct->diodrvr_msec = msec;
        ct->diodrvr_usec = usec;
        ct->diodrvr_nsec = nsec;

}
```

E. Framebuffer schematics

The following pages contain the electronic schematics of the MVD framebuffer electronics.

SLM INTERFACE MAIN BOARD BLOCK DIAGRAM



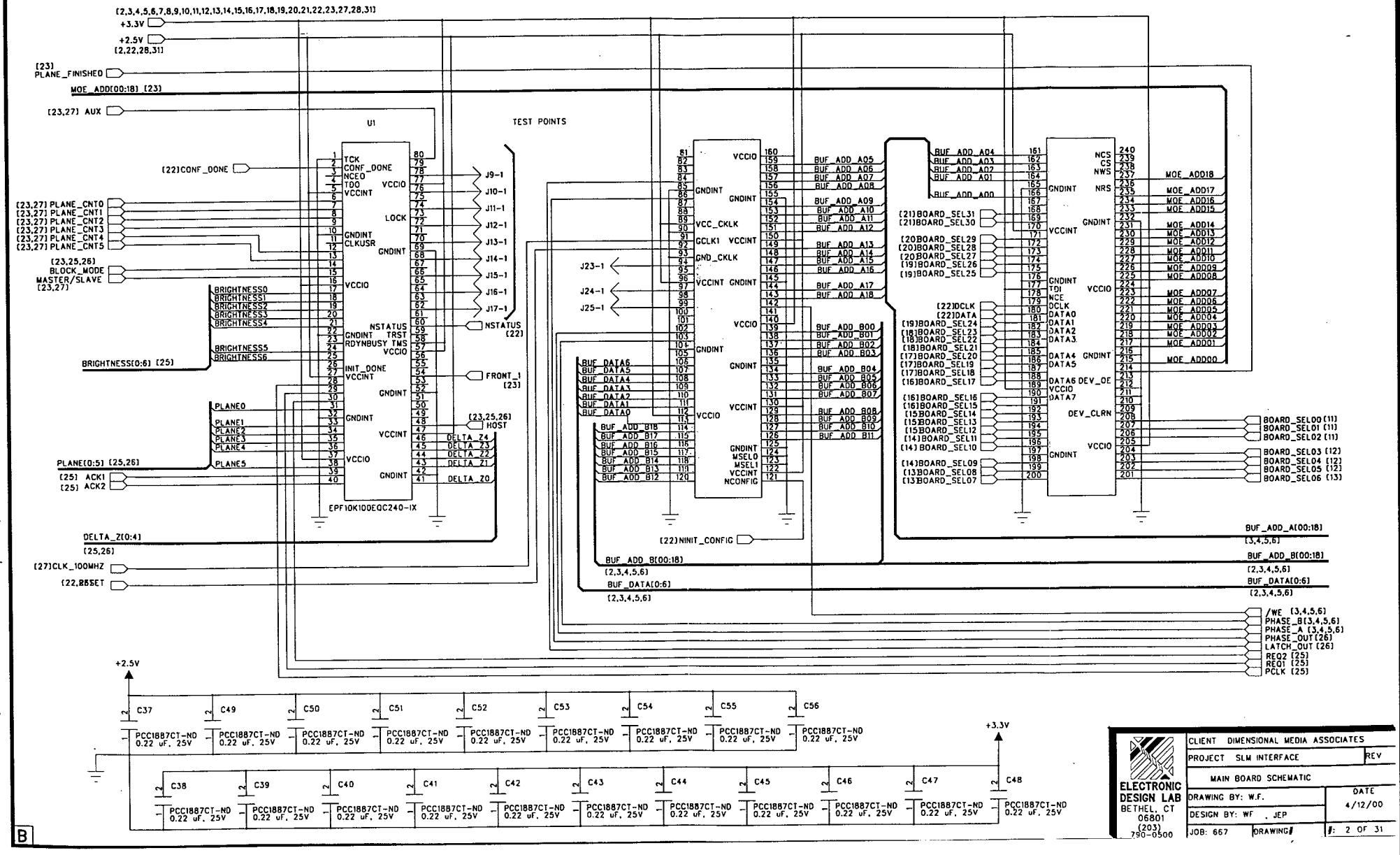
NOTES:

- SRAM CARDS PLUG INTO MAIN BOARD.
- SRAM CARD CONNECTORS ARE ON PAGES 11 TROUGH 21.
- BRACES "[ ]" ENCLOSE PAGE NUMBERS.
- CIRCLES "◯" ARE BNC CONNECTORS.
- [ ] ENCLOSE PAGE NUMBERS.
- RAMDAC VAA BYPASS CAPACITORS ARE ON PAGES 29 AND 30.

ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500

CLIENT DIMENSIONAL MEDIA ASSOCIATES		
PROJECT SLM INTERFACE		REV
MAIN BOARD SCHEMATIC		
DRAWING BY: W.F.		DATE 4/12/00
DESIGN BY: WF , JEP		
JOB: 667	DRAWING#	#: 1 OF 31

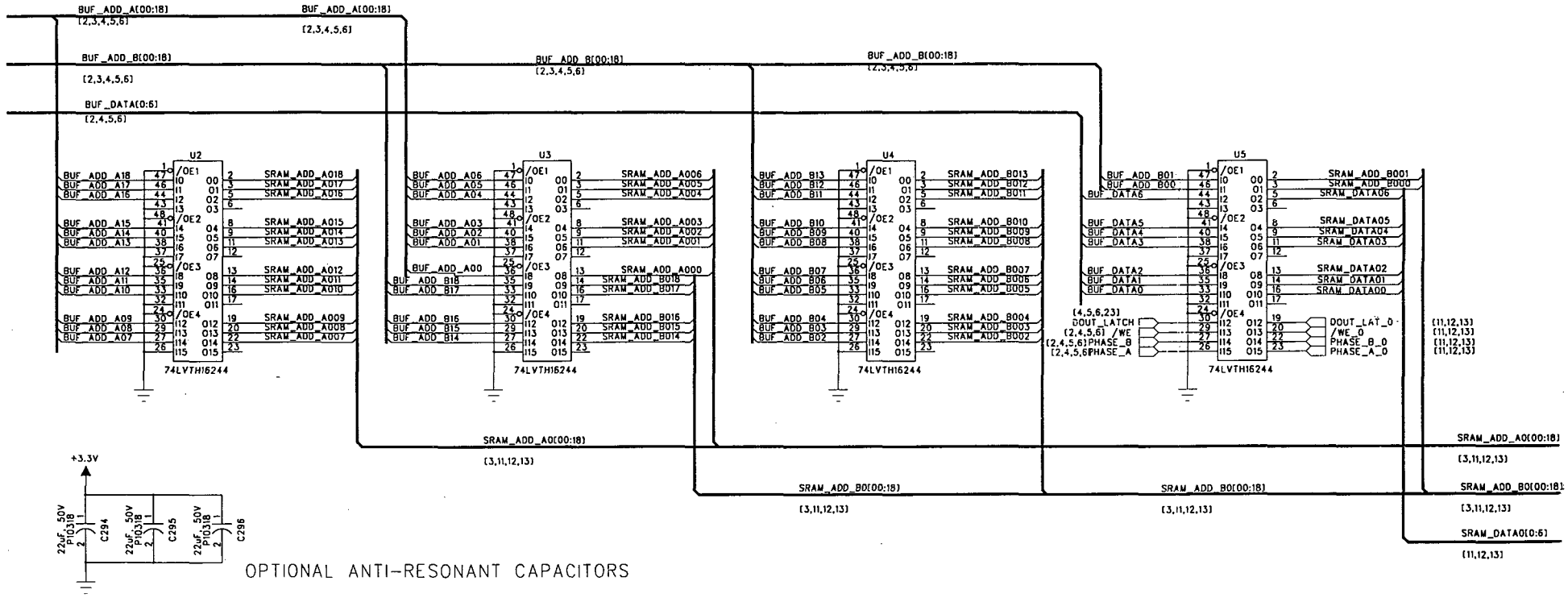
FRONT. END LOGIC



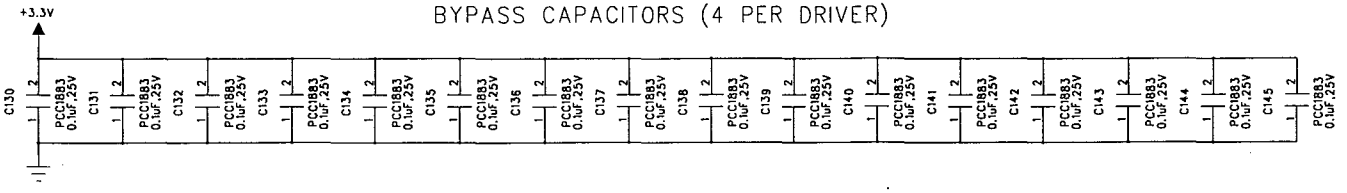
B

ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING#: 2 OF 31

BUS BUFFER GROUP 0



OPTIONAL ANTI-RESONANT CAPACITORS

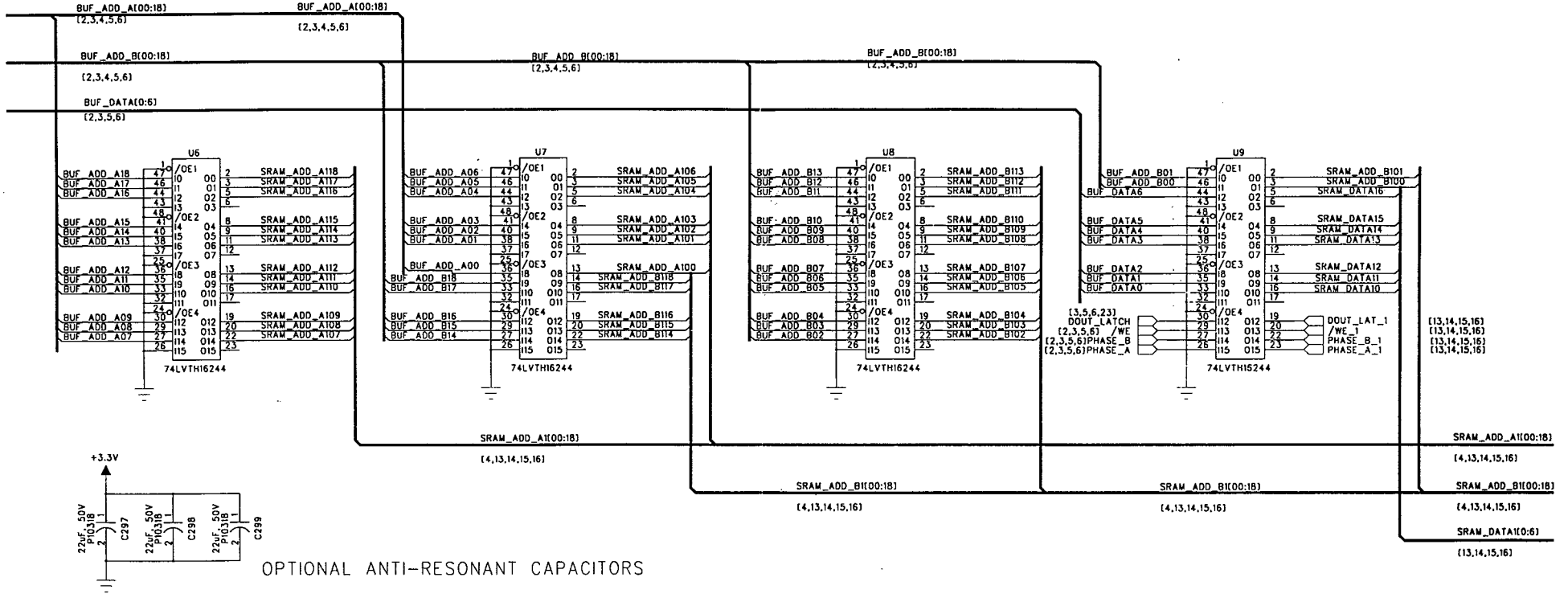


BYPASS CAPACITORS (4 PER DRIVER)

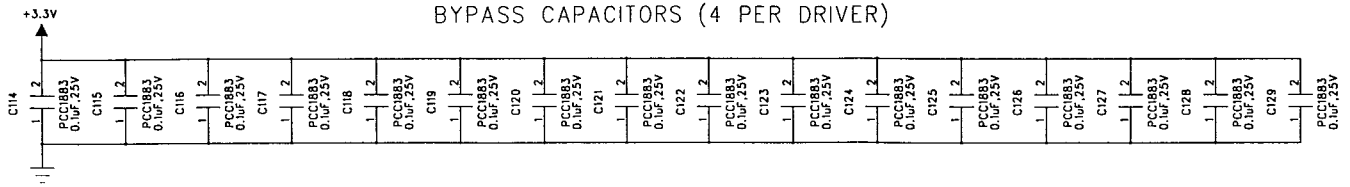
ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING#: 3 OF 31

B

BUS BUFFER GROUP 1



OPTIONAL ANTI-RESONANT CAPACITORS

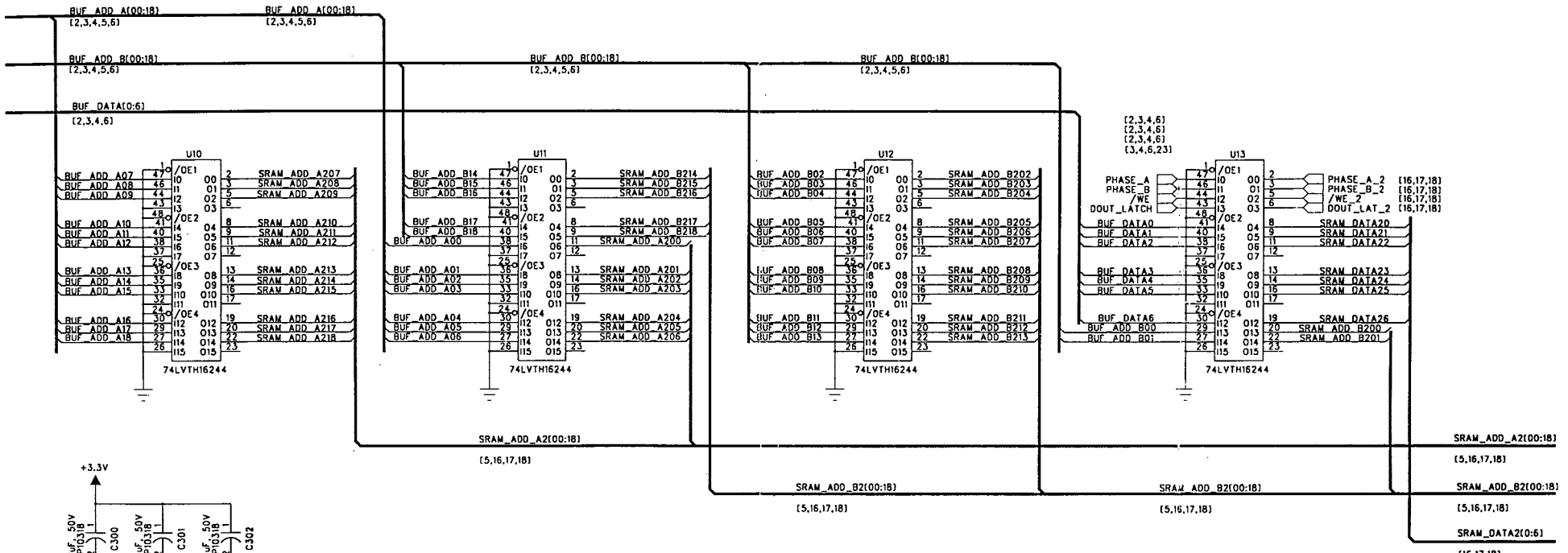


BYPASS CAPACITORS (4 PER DRIVER)

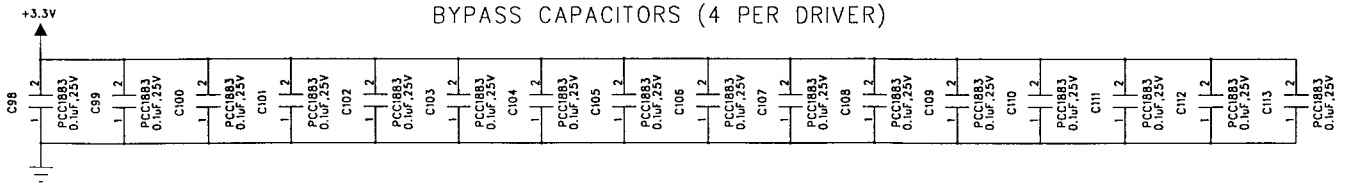
ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING#: — #: 4 OF 31

B

BUS BUFFER GROUP 2



OPTIONAL ANTI-RESONANT CAPACITORS

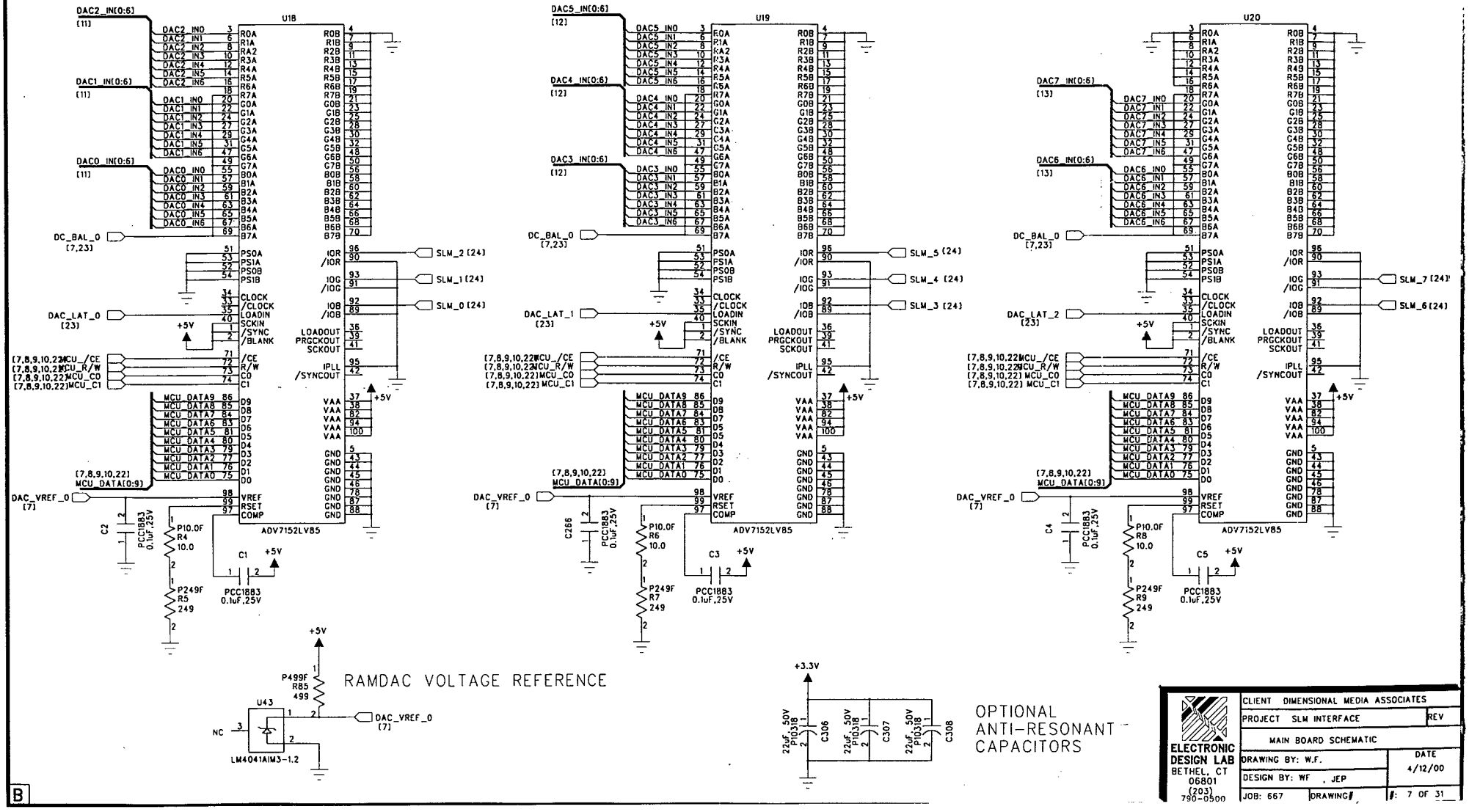


BYPASS CAPACITORS (4 PER DRIVER)

ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203)	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING# — #: 5 OF 31

B

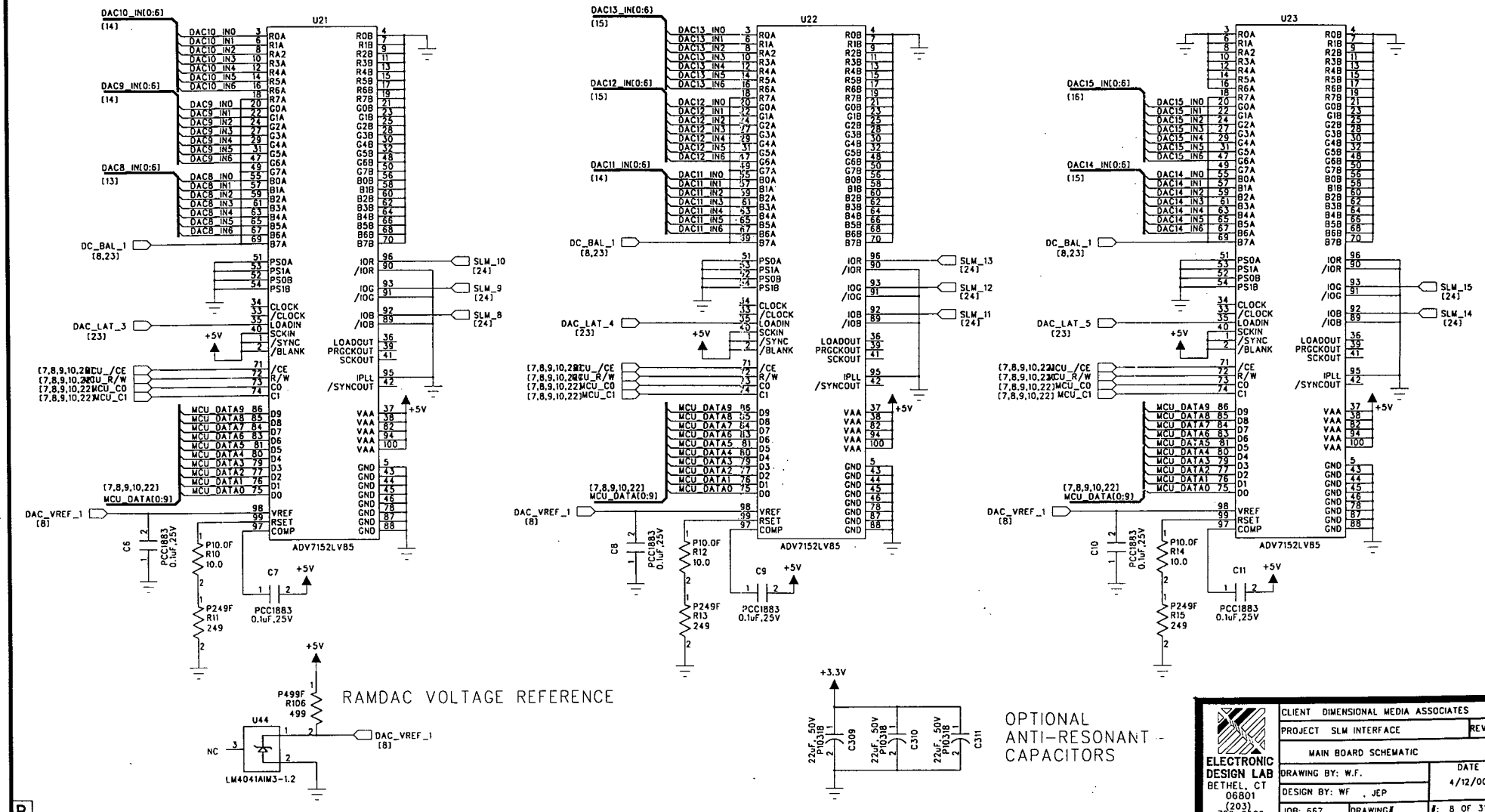
GROUP 0 RAMDACS



CLIENT	DIMENSIONAL MEDIA ASSOCIATES
PROJECT	SLM INTERFACE
REV	
	MAIN BOARD SCHEMATIC
DRAWING BY:	W.F.
DESIGN BY:	WF , JEP
DATE	4/12/00
JOB:	667
DRAWING #	7 OF 31

ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500

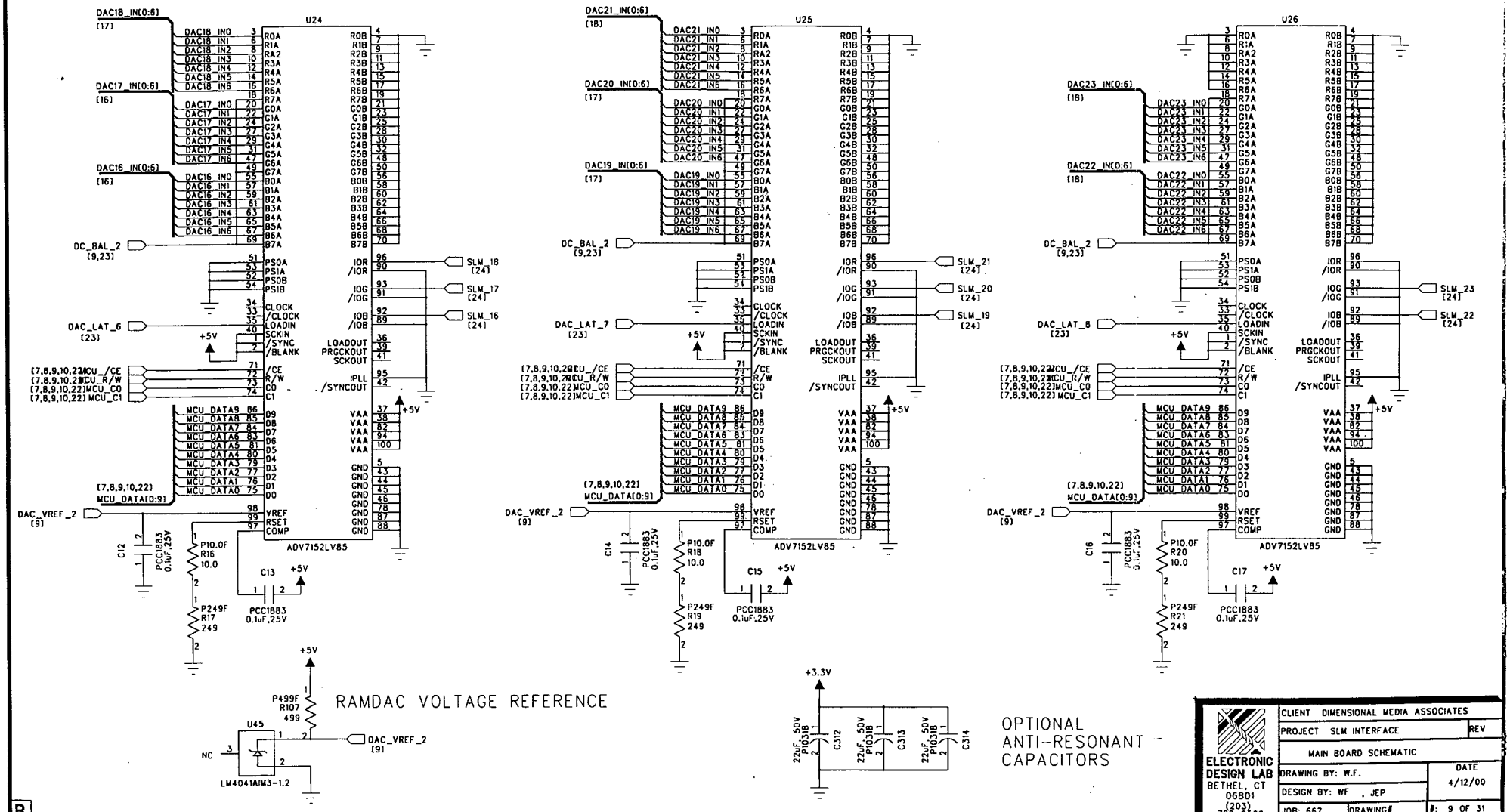
GROUP 1 RAMDACS



ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING# — #: 8 OF 31

B

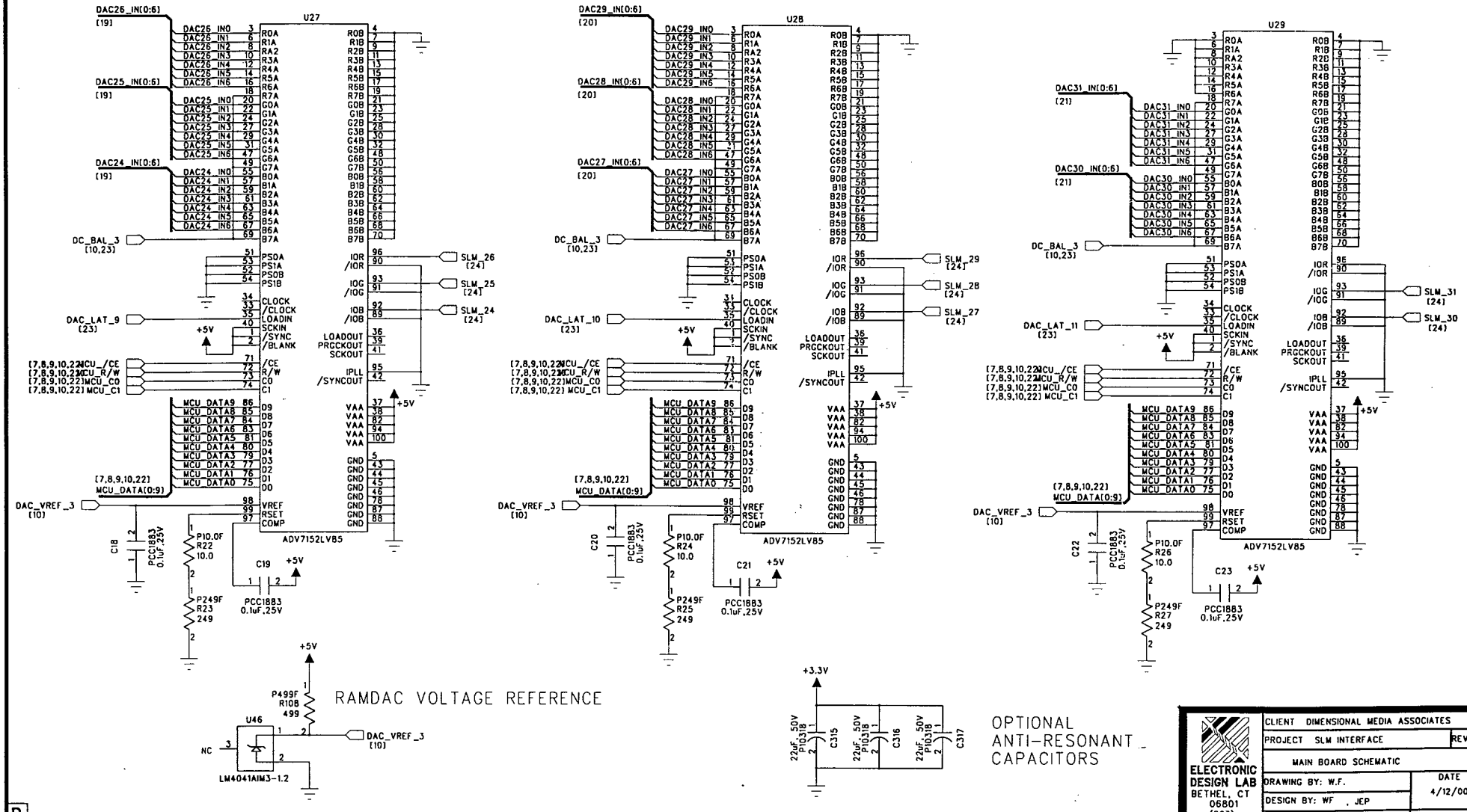
GROUP 2 RAMDACS



ELECTRONIC DESIGN LAB, BETHEL, CT 06801, (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING#: — #: 9 OF 31

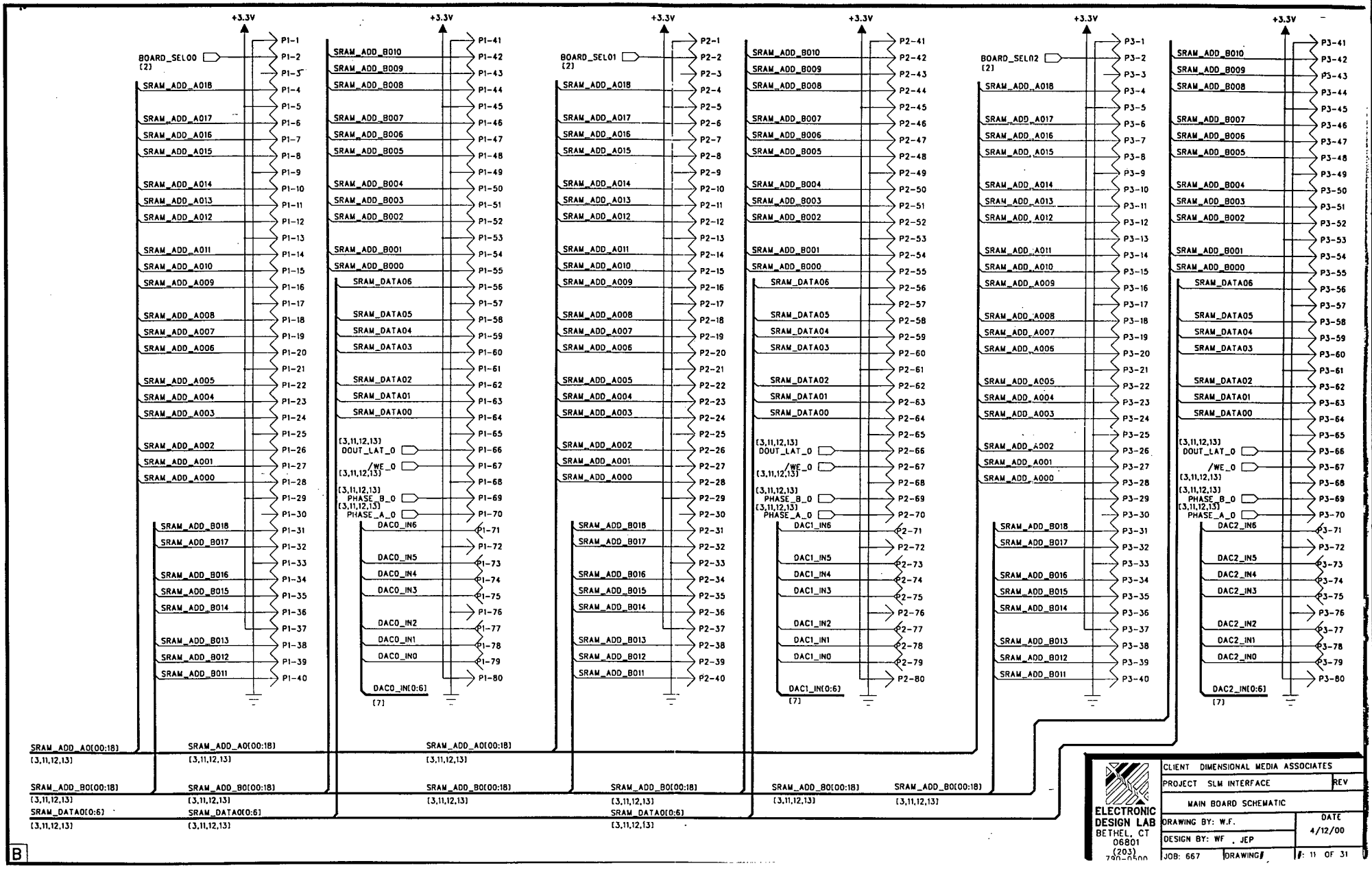
B

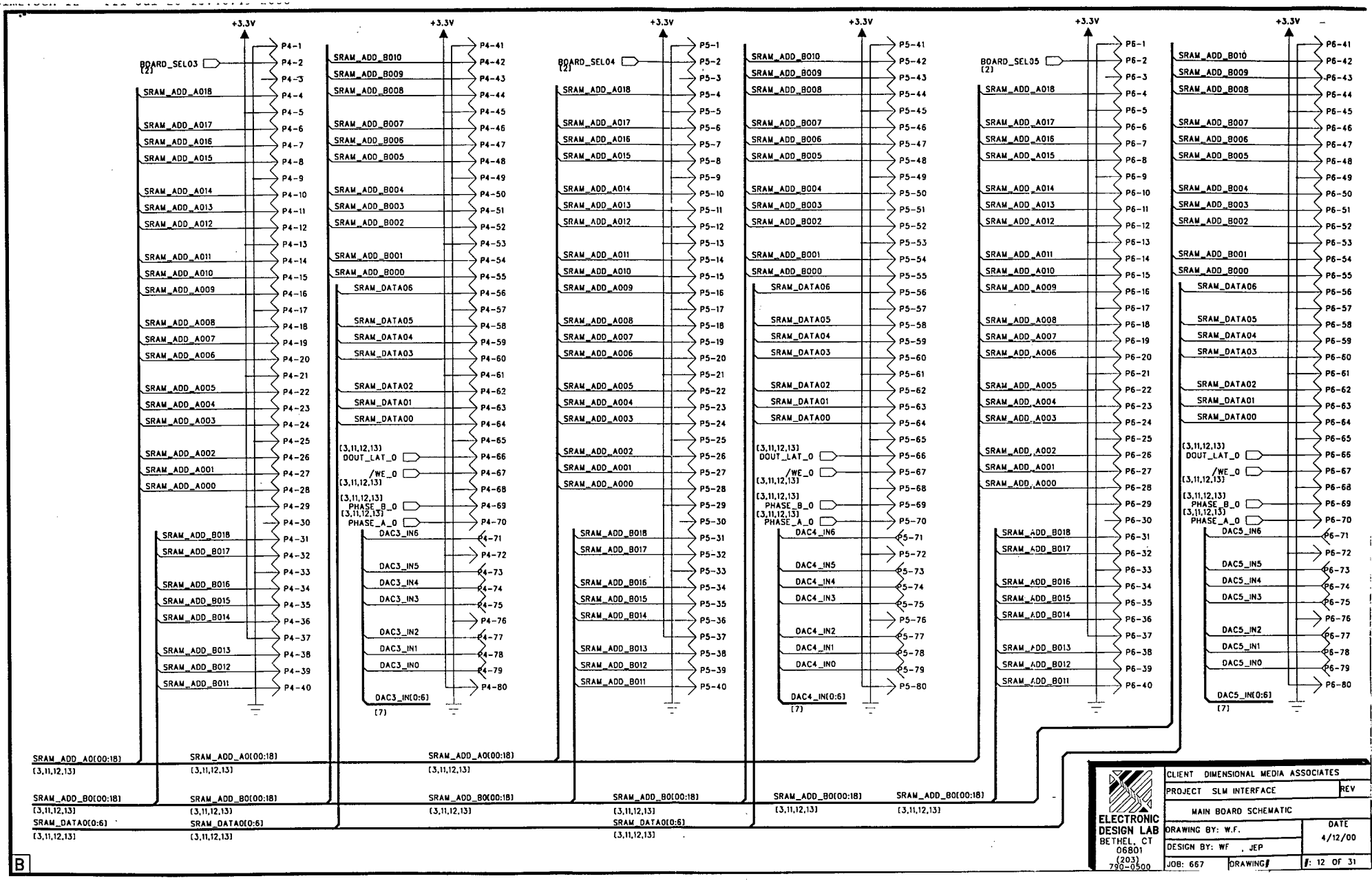
GROUP 3 RAMDACS

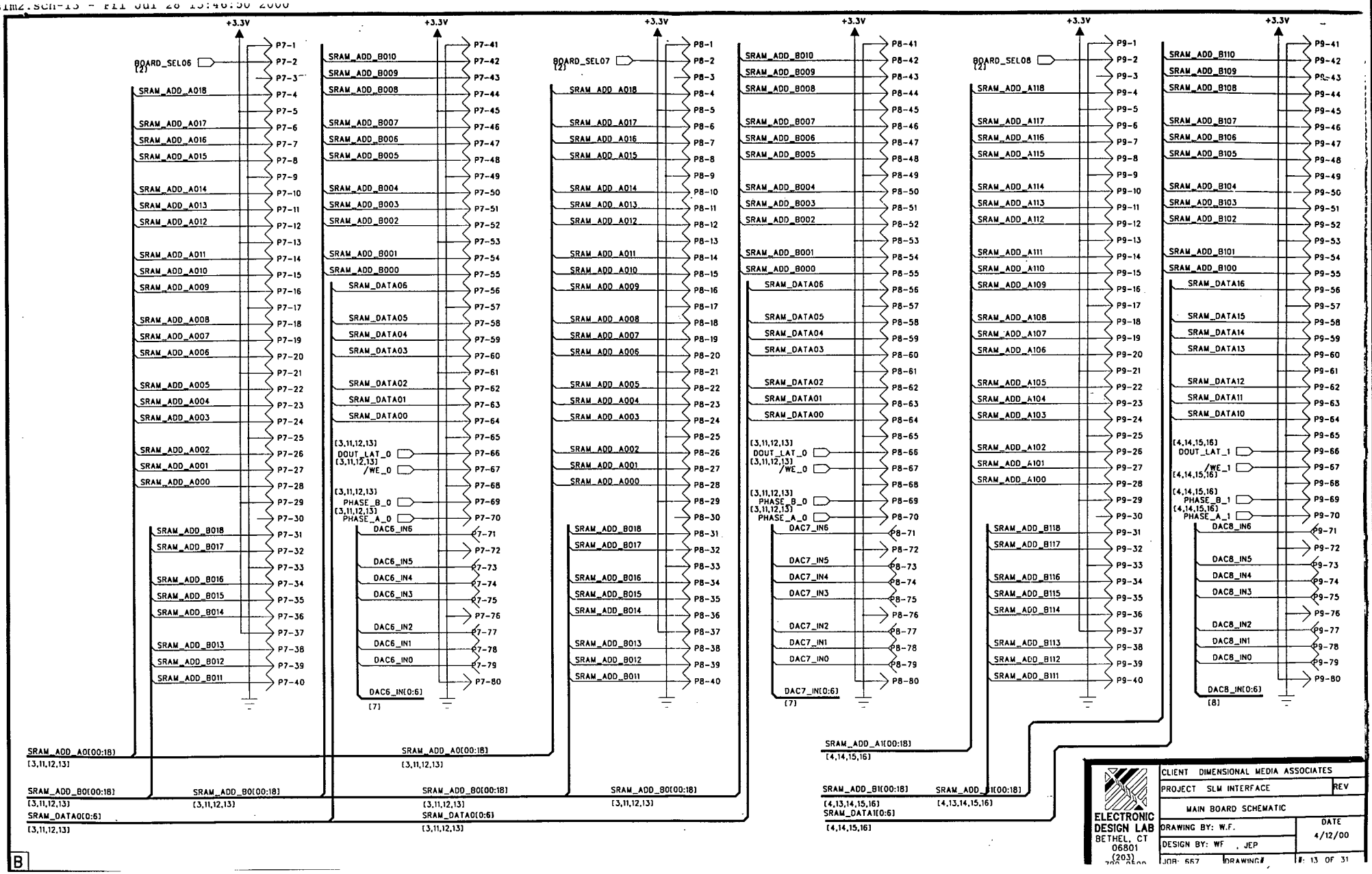


ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING# — #: 10 OF 31

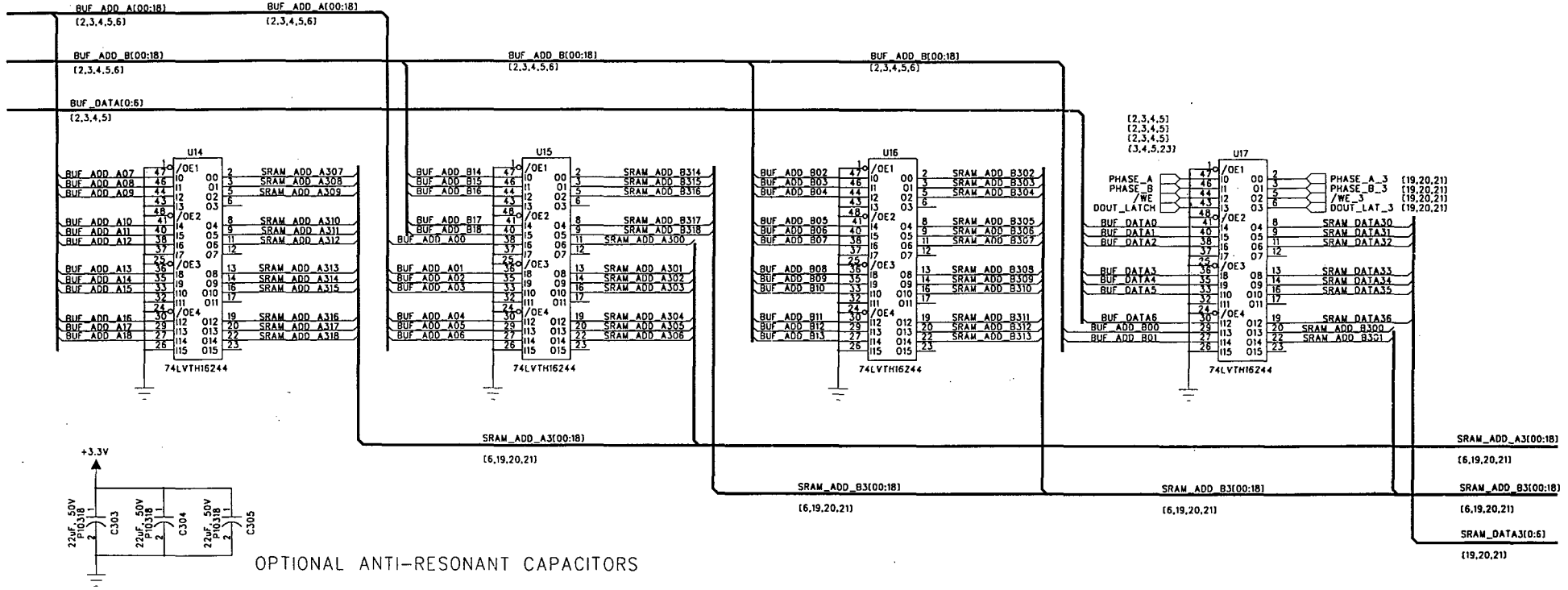
B


+3.3V
BOARD_SEL00
[2]
BOARD_SEL01
BOARD_SEL02
SRAM_ADD_A018
SRAM_ADD_A017
SRAM_ADD_A016
SRAM_ADD_A015
SRAM_ADD_A014
SRAM_ADD_A013
SRAM_ADD_A012
SRAM_ADD_A011
SRAM_ADD_A010
SRAM_ADD_A009
SRAM_ADD_A008
SRAM_ADD_A007
SRAM_ADD_A006
SRAM_ADD_A005
SRAM_ADD_A004
SRAM_ADD_A003
SRAM_ADD_A002
SRAM_ADD_A001
SRAM_ADD_A000
SRAM_ADD_B018
SRAM_ADD_B017
SRAM_ADD_B016
SRAM_ADD_B015
SRAM_ADD_B014
SRAM_ADD_B013
SRAM_ADD_B012
SRAM_ADD_B011
SRAM_ADD_B010
SRAM_ADD_B009
SRAM_ADD_B008
SRAM_ADD_B007
SRAM_ADD_B006
SRAM_ADD_B005
SRAM_ADD_B004
SRAM_ADD_B003
SRAM_ADD_B002
SRAM_ADD_B001
SRAM_ADD_B000
SRAM_DATA06
SRAM_DATA05
SRAM_DATA04
SRAM_DATA03
SRAM_DATA02
SRAM_DATA01
SRAM_DATA00
[3,11,12,13]
DOUT_LAT_0
/WE_0
PHASE_B_0
PHASE_A_0
DAC0_IN6
DAC0_IN5
DAC0_IN4
DAC0_IN3
DAC0_IN2
DAC0_IN1
DAC0_IN0
DAC0_IN[0:6]
[7]
DAC1_IN6
DAC1_IN5
DAC1_IN4
DAC1_IN3
DAC1_IN2
DAC1_IN1
DAC1_IN0
DAC1_IN[0:6]
DAC2_IN6
DAC2_IN5
DAC2_IN4
DAC2_IN3
DAC2_IN2
DAC2_IN1
DAC2_IN0
DAC2_IN[0:6]
SRAM_ADD_A0[00:18]
[3,11,12,13]
SRAM_ADD_B0[00:18]
SRAM_DATA0[0:6]
B
ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE
4/12/00
JOB: 667
DRAWING #
#: 11 OF 31

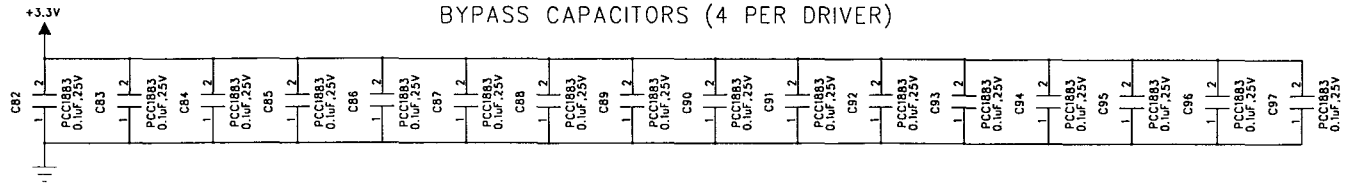

+3.3V
BOARD_SEL03
BOARD_SEL04
BOARD_SEL05
SRAM_ADD_A018
SRAM_ADD_A017
SRAM_ADD_A016
SRAM_ADD_A015
SRAM_ADD_A014
SRAM_ADD_A013
SRAM_ADD_A012
SRAM_ADD_A011
SRAM_ADD_A010
SRAM_ADD_A009
SRAM_ADD_A008
SRAM_ADD_A007
SRAM_ADD_A006
SRAM_ADD_A005
SRAM_ADD_A004
SRAM_ADD_A003
SRAM_ADD_A002
SRAM_ADD_A001
SRAM_ADD_A000
SRAM_ADD_B018
SRAM_ADD_B017
SRAM_ADD_B016
SRAM_ADD_B015
SRAM_ADD_B014
SRAM_ADD_B013
SRAM_ADD_B012
SRAM_ADD_B011
SRAM_ADD_B010
SRAM_ADD_B009
SRAM_ADD_B008
SRAM_ADD_B007
SRAM_ADD_B006
SRAM_ADD_B005
SRAM_ADD_B004
SRAM_ADD_B003
SRAM_ADD_B002
SRAM_ADD_B001
SRAM_ADD_B000
SRAM_DATA06
SRAM_DATA05
SRAM_DATA04
SRAM_DATA03
SRAM_DATA02
SRAM_DATA01
SRAM_DATA00
DOUT_LAT_0
/WE_0
PHASE_B_0
PHASE_A_0
DAC3_IN6
DAC3_IN5
DAC3_IN4
DAC3_IN3
DAC3_IN2
DAC3_IN1
DAC3_IN0
DAC3_IN[0:6]
DAC4_IN6
DAC4_IN5
DAC4_IN4
DAC4_IN3
DAC4_IN2
DAC4_IN1
DAC4_IN0
DAC4_IN[0:6]
DAC5_IN6
DAC5_IN5
DAC5_IN4
DAC5_IN3
DAC5_IN2
DAC5_IN1
DAC5_IN0
DAC5_IN[0:6]
SRAM_ADD_A[00:18]
SRAM_ADD_B[00:18]
SRAM_DATA[0:6]
ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203) 790-0500
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE 4/12/00
JOB: 667
DRAWING #: 12 OF 31


+3.3V
BOARD_SEL06 [2]
SRAM_ADD_A018
SRAM_ADD_A017
SRAM_ADD_A016
SRAM_ADD_A015
SRAM_ADD_A014
SRAM_ADD_A013
SRAM_ADD_A012
SRAM_ADD_A011
SRAM_ADD_A010
SRAM_ADD_A009
SRAM_ADD_A008
SRAM_ADD_A007
SRAM_ADD_A006
SRAM_ADD_A005
SRAM_ADD_A004
SRAM_ADD_A003
SRAM_ADD_A002
SRAM_ADD_A001
SRAM_ADD_A000
SRAM_ADD_B018
SRAM_ADD_B017
SRAM_ADD_B016
SRAM_ADD_B015
SRAM_ADD_B014
SRAM_ADD_B013
SRAM_ADD_B012
SRAM_ADD_B011
P7-1 P7-2 P7-3 P7-4 P7-5 P7-6 P7-7 P7-8 P7-9 P7-10 P7-11 P7-12 P7-13 P7-14 P7-15 P7-16 P7-17 P7-18 P7-19 P7-20 P7-21 P7-22 P7-23 P7-24 P7-25 P7-26 P7-27 P7-28 P7-29 P7-30 P7-31 P7-32 P7-33 P7-34 P7-35 P7-36 P7-37 P7-38 P7-39 P7-40
SRAM_ADD_B010
SRAM_ADD_B009
SRAM_ADD_B008
SRAM_ADD_B007
SRAM_ADD_B006
SRAM_ADD_B005
SRAM_ADD_B004
SRAM_ADD_B003
SRAM_ADD_B002
SRAM_ADD_B001
SRAM_ADD_B000
SRAM_DATA06
SRAM_DATA05
SRAM_DATA04
SRAM_DATA03
SRAM_DATA02
SRAM_DATA01
SRAM_DATA00
[3,11,12,13] DOUT_LAT_0
[3,11,12,13] /WE_0
[3,11,12,13] PHASE_B_0
[3,11,12,13] PHASE_A_0
DAC6_IN6
DAC6_IN5
DAC6_IN4
DAC6_IN3
DAC6_IN2
DAC6_IN1
DAC6_IN0
DAC6_IN[0:6] [7]
P7-41 P7-42 P7-43 P7-44 P7-45 P7-46 P7-47 P7-48 P7-49 P7-50 P7-51 P7-52 P7-53 P7-54 P7-55 P7-56 P7-57 P7-58 P7-59 P7-60 P7-61 P7-62 P7-63 P7-64 P7-65 P7-66 P7-67 P7-68 P7-69 P7-70 P7-71 P7-72 P7-73 P7-74 P7-75 P7-76 P7-77 P7-78 P7-79 P7-80
BOARD_SEL07 [2]
P8-1 P8-2 P8-3 P8-4 P8-5 P8-6 P8-7 P8-8 P8-9 P8-10 P8-11 P8-12 P8-13 P8-14 P8-15 P8-16 P8-17 P8-18 P8-19 P8-20 P8-21 P8-22 P8-23 P8-24 P8-25 P8-26 P8-27 P8-28 P8-29 P8-30 P8-31 P8-32 P8-33 P8-34 P8-35 P8-36 P8-37 P8-38 P8-39 P8-40
P8-41 P8-42 P8-43 P8-44 P8-45 P8-46 P8-47 P8-48 P8-49 P8-50 P8-51 P8-52 P8-53 P8-54 P8-55 P8-56 P8-57 P8-58 P8-59 P8-60 P8-61 P8-62 P8-63 P8-64 P8-65 P8-66 P8-67 P8-68 P8-69 P8-70 P8-71 P8-72 P8-73 P8-74 P8-75 P8-76 P8-77 P8-78 P8-79 P8-80
DAC7_IN6
DAC7_IN5
DAC7_IN4
DAC7_IN3
DAC7_IN2
DAC7_IN1
DAC7_IN0
DAC7_IN[0:6] [7]
BOARD_SEL08 [2]
P9-1 P9-2 P9-3 P9-4 P9-5 P9-6 P9-7 P9-8 P9-9 P9-10 P9-11 P9-12 P9-13 P9-14 P9-15 P9-16 P9-17 P9-18 P9-19 P9-20 P9-21 P9-22 P9-23 P9-24 P9-25 P9-26 P9-27 P9-28 P9-29 P9-30 P9-31 P9-32 P9-33 P9-34 P9-35 P9-36 P9-37 P9-38 P9-39 P9-40
SRAM_ADD_A118
SRAM_ADD_A117
SRAM_ADD_A116
SRAM_ADD_A115
SRAM_ADD_A114
SRAM_ADD_A113
SRAM_ADD_A112
SRAM_ADD_A111
SRAM_ADD_A110
SRAM_ADD_A109
SRAM_ADD_A108
SRAM_ADD_A107
SRAM_ADD_A106
SRAM_ADD_A105
SRAM_ADD_A104
SRAM_ADD_A103
SRAM_ADD_A102
SRAM_ADD_A101
SRAM_ADD_A100
SRAM_ADD_B118
SRAM_ADD_B117
SRAM_ADD_B116
SRAM_ADD_B115
SRAM_ADD_B114
SRAM_ADD_B113
SRAM_ADD_B112
SRAM_ADD_B111
P9-41 P9-42 P9-43 P9-44 P9-45 P9-46 P9-47 P9-48 P9-49 P9-50 P9-51 P9-52 P9-53 P9-54 P9-55 P9-56 P9-57 P9-58 P9-59 P9-60 P9-61 P9-62 P9-63 P9-64 P9-65 P9-66 P9-67 P9-68 P9-69 P9-70 P9-71 P9-72 P9-73 P9-74 P9-75 P9-76 P9-77 P9-78 P9-79 P9-80
SRAM_ADD_B110
SRAM_ADD_B109
SRAM_ADD_B108
SRAM_ADD_B107
SRAM_ADD_B106
SRAM_ADD_B105
SRAM_ADD_B104
SRAM_ADD_B103
SRAM_ADD_B102
SRAM_ADD_B101
SRAM_ADD_B100
SRAM_DATA16
SRAM_DATA15
SRAM_DATA14
SRAM_DATA13
SRAM_DATA12
SRAM_DATA11
SRAM_DATA10
[4,14,15,16] DOUT_LAT_1
[4,14,15,16] /WE_1
[4,14,15,16] PHASE_B_1
[4,14,15,16] PHASE_A_1
DAC8_IN6
DAC8_IN5
DAC8_IN4
DAC8_IN3
DAC8_IN2
DAC8_IN1
DAC8_IN0
DAC8_IN[0:6] [8]
SRAM_ADD_A0[00:18] [3,11,12,13]
SRAM_ADD_B0[00:18] [3,11,12,13]
SRAM_DATA0[0:6] [3,11,12,13]
SRAM_ADD_A1[00:18] [4,14,15,16]
SRAM_ADD_B1[00:18] [4,13,14,15,16]
SRAM_DATA1[0:6] [4,14,15,16]
B
ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203)
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE 4/12/00
JOB: 667
DRAWING#
#: 13 OF 31

BUS BUFFER GROUP 3



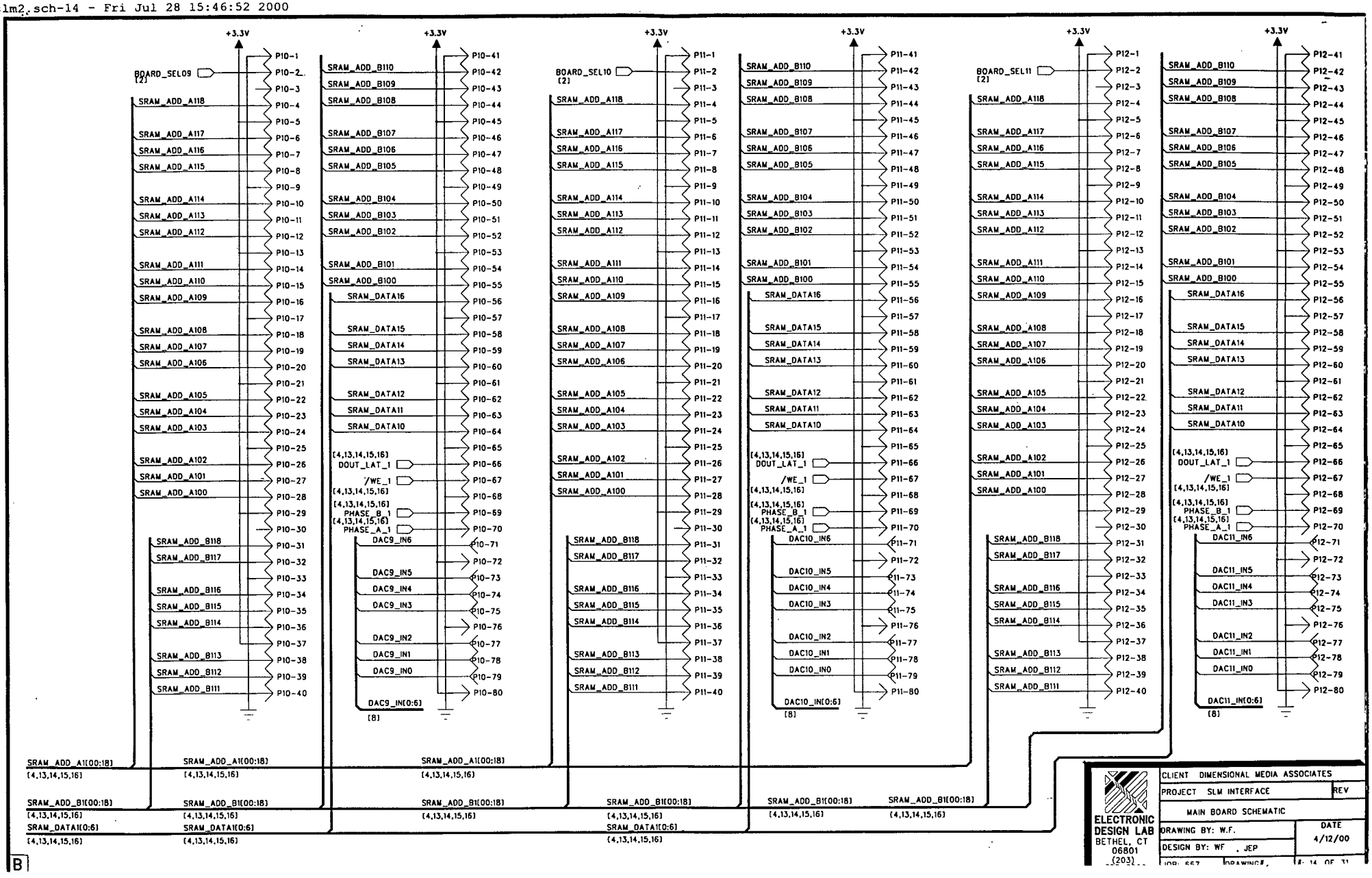
OPTIONAL ANTI-RESONANT CAPACITORS

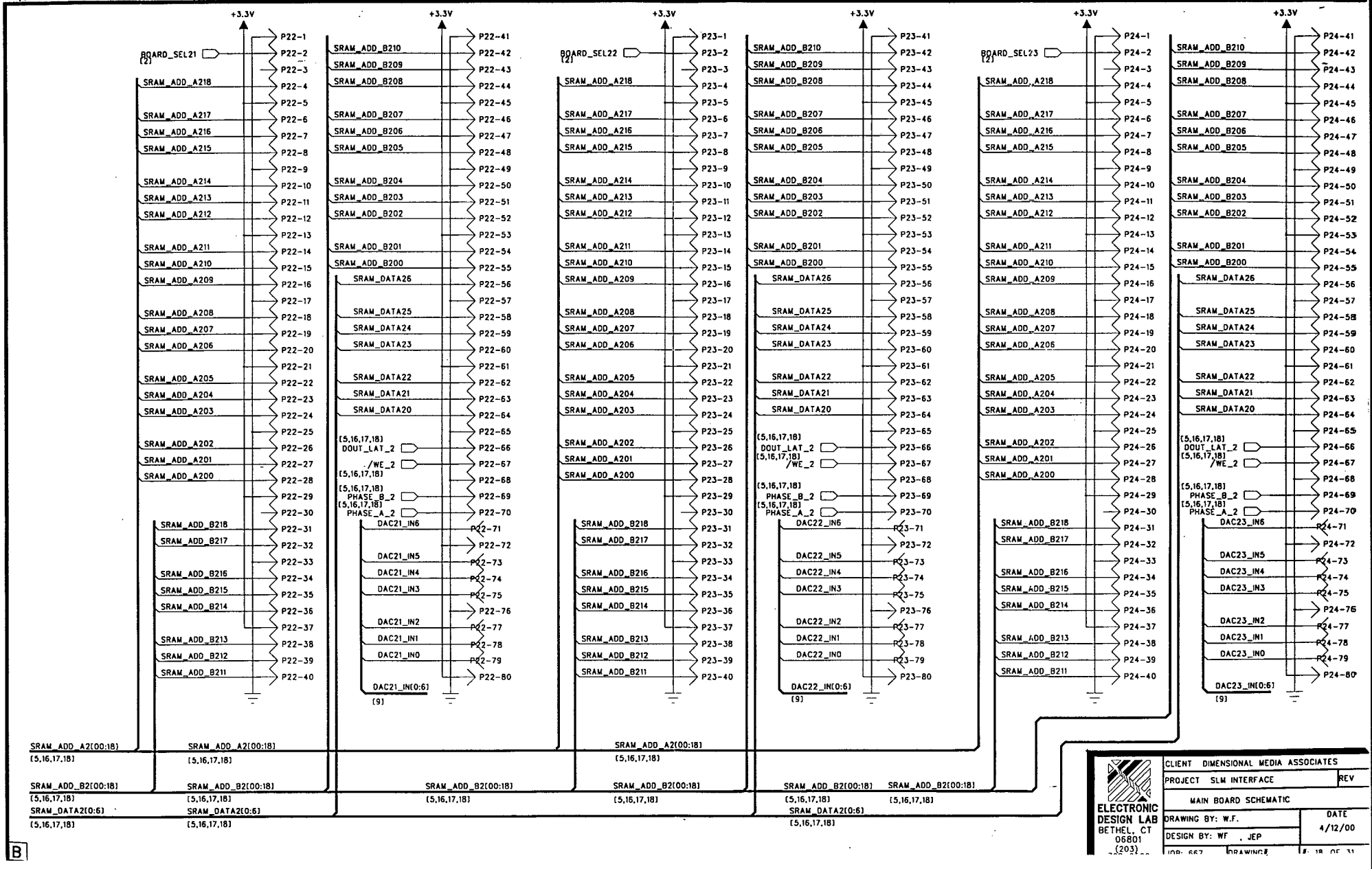


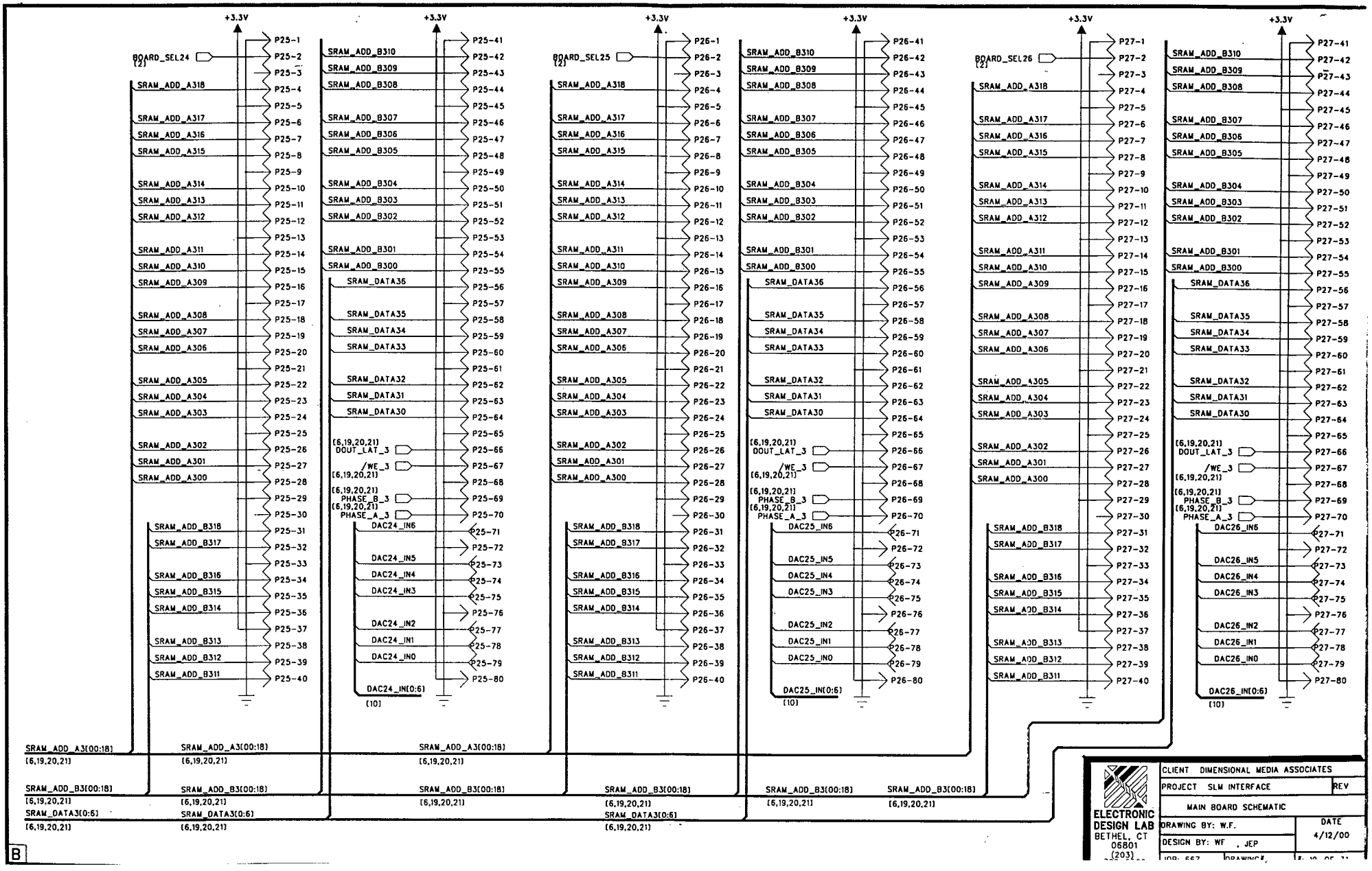
BYPASS CAPACITORS (4 PER DRIVER)

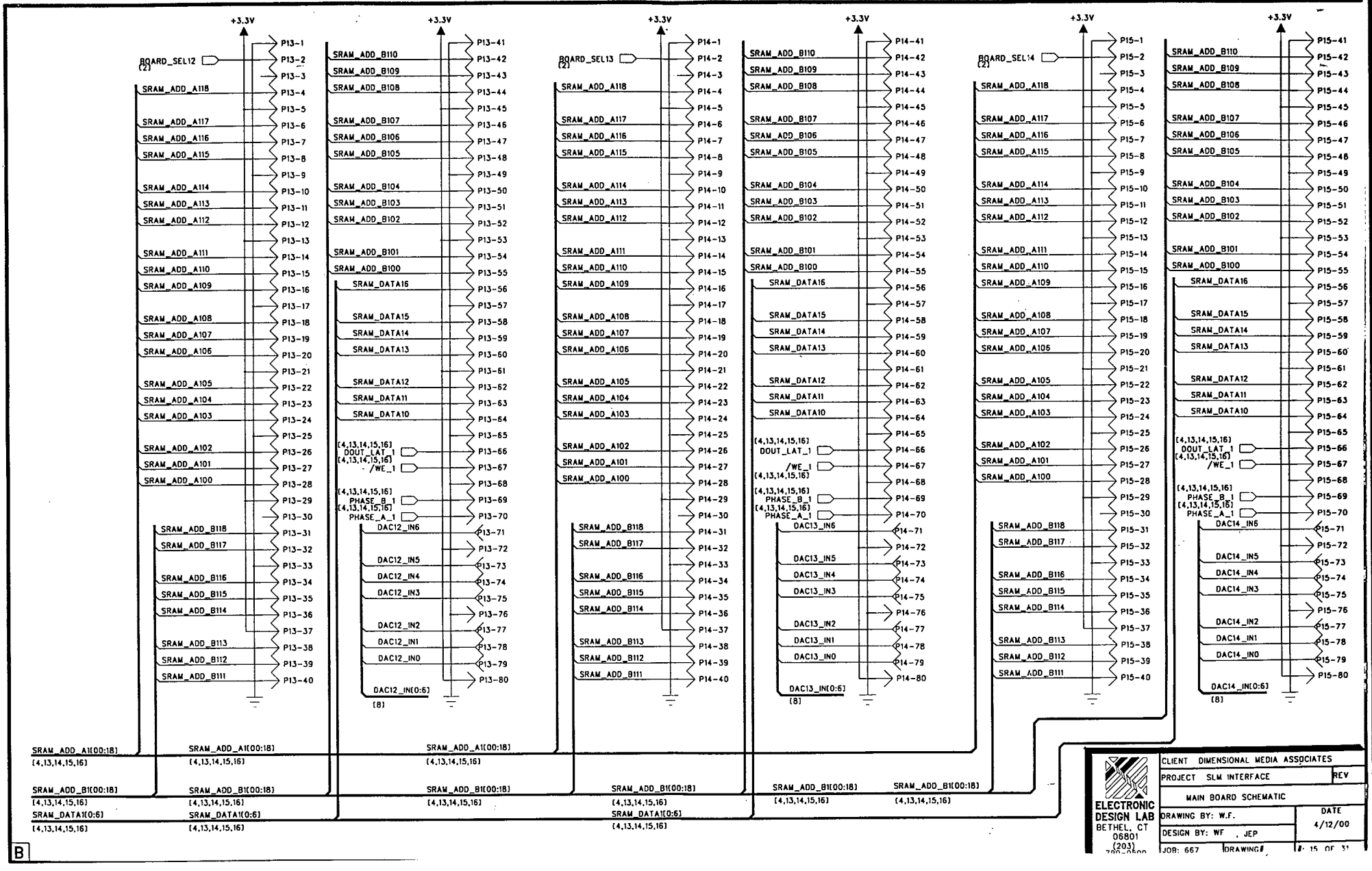
ELECTRONIC DESIGN LAB, BETHEL, CT 06801, (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#	#: 6 OF 31

B

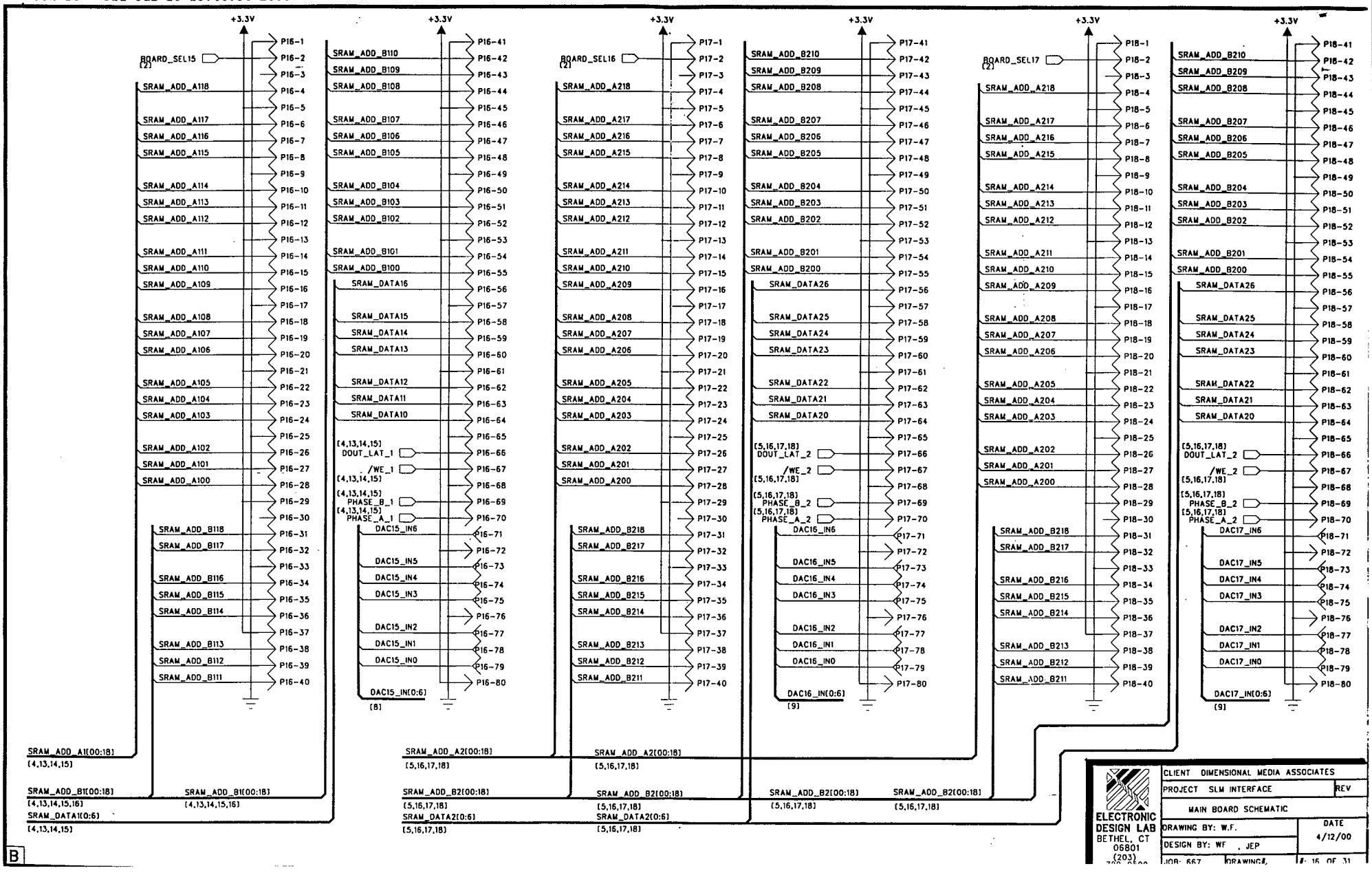

+3.3V
BOARD_SEL09
BOARD_SEL10
BOARD_SEL11
SRAM_ADD_A118
SRAM_ADD_A117
SRAM_ADD_A116
SRAM_ADD_A115
SRAM_ADD_A114
SRAM_ADD_A113
SRAM_ADD_A112
SRAM_ADD_A111
SRAM_ADD_A110
SRAM_ADD_A109
SRAM_ADD_A108
SRAM_ADD_A107
SRAM_ADD_A106
SRAM_ADD_A105
SRAM_ADD_A104
SRAM_ADD_A103
SRAM_ADD_A102
SRAM_ADD_A101
SRAM_ADD_A100
SRAM_ADD_B118
SRAM_ADD_B117
SRAM_ADD_B116
SRAM_ADD_B115
SRAM_ADD_B114
SRAM_ADD_B113
SRAM_ADD_B112
SRAM_ADD_B111
SRAM_ADD_B110
SRAM_ADD_B109
SRAM_ADD_B108
SRAM_ADD_B107
SRAM_ADD_B106
SRAM_ADD_B105
SRAM_ADD_B104
SRAM_ADD_B103
SRAM_ADD_B102
SRAM_ADD_B101
SRAM_ADD_B100
SRAM_DATA16
SRAM_DATA15
SRAM_DATA14
SRAM_DATA13
SRAM_DATA12
SRAM_DATA11
SRAM_DATA10
DOUT_LAT_1
/WE_1
PHASE_B_1
PHASE_A_1
DAC9_IN6
DAC9_IN5
DAC9_IN4
DAC9_IN3
DAC9_IN2
DAC9_IN1
DAC9_IN0
DAC9_IN[0:6]
DAC10_IN6
DAC10_IN5
DAC10_IN4
DAC10_IN3
DAC10_IN2
DAC10_IN1
DAC10_IN0
DAC10_IN[0:6]
DAC11_IN6
DAC11_IN5
DAC11_IN4
DAC11_IN3
DAC11_IN2
DAC11_IN1
DAC11_IN0
DAC11_IN[0:6]
SRAM_ADD_A1[00:18]
SRAM_ADD_B1[00:18]
SRAM_DATA1[0:6]
ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203)
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE 4/12/00


+3.3V
BOARD_SEL21
(2)
P22-1
P22-2
P22-3
P22-4
P22-5
P22-6
P22-7
P22-8
P22-9
P22-10
P22-11
P22-12
P22-13
P22-14
P22-15
P22-16
P22-17
P22-18
P22-19
P22-20
P22-21
P22-22
P22-23
P22-24
P22-25
P22-26
P22-27
P22-28
P22-29
P22-30
P22-31
P22-32
P22-33
P22-34
P22-35
P22-36
P22-37
P22-38
P22-39
P22-40
SRAM_ADD_A218
SRAM_ADD_A217
SRAM_ADD_A216
SRAM_ADD_A215
SRAM_ADD_A214
SRAM_ADD_A213
SRAM_ADD_A212
SRAM_ADD_A211
SRAM_ADD_A210
SRAM_ADD_A209
SRAM_ADD_A208
SRAM_ADD_A207
SRAM_ADD_A206
SRAM_ADD_A205
SRAM_ADD_A204
SRAM_ADD_A203
SRAM_ADD_A202
SRAM_ADD_A201
SRAM_ADD_A200
SRAM_ADD_B218
SRAM_ADD_B217
SRAM_ADD_B216
SRAM_ADD_B215
SRAM_ADD_B214
SRAM_ADD_B213
SRAM_ADD_B212
SRAM_ADD_B211
+3.3V
P22-41
P22-42
P22-43
P22-44
P22-45
P22-46
P22-47
P22-48
P22-49
P22-50
P22-51
P22-52
P22-53
P22-54
P22-55
P22-56
P22-57
P22-58
P22-59
P22-60
P22-61
P22-62
P22-63
P22-64
P22-65
P22-66
P22-67
P22-68
P22-69
P22-70
P22-71
P22-72
P22-73
P22-74
P22-75
P22-76
P22-77
P22-78
P22-79
P22-80
SRAM_ADD_B210
SRAM_ADD_B209
SRAM_ADD_B208
SRAM_ADD_B207
SRAM_ADD_B206
SRAM_ADD_B205
SRAM_ADD_B204
SRAM_ADD_B203
SRAM_ADD_B202
SRAM_ADD_B201
SRAM_ADD_B200
SRAM_DATA26
SRAM_DATA25
SRAM_DATA24
SRAM_DATA23
SRAM_DATA22
SRAM_DATA21
SRAM_DATA20
[5,16,17,18]
DOUT_LAT_2
./WE_2
[5,16,17,18]
[5,16,17,18]
PHASE_B_2
[5,16,17,18]
PHASE_A_2
DAC21_IN6
DAC21_IN5
DAC21_IN4
DAC21_IN3
DAC21_IN2
DAC21_IN1
DAC21_IN0
DAC21_IN[0:6]
[9]
BOARD_SEL22
(2)
P23-1
P23-2
P23-3
P23-4
P23-5
P23-6
P23-7
P23-8
P23-9
P23-10
P23-11
P23-12
P23-13
P23-14
P23-15
P23-16
P23-17
P23-18
P23-19
P23-20
P23-21
P23-22
P23-23
P23-24
P23-25
P23-26
P23-27
P23-28
P23-29
P23-30
P23-31
P23-32
P23-33
P23-34
P23-35
P23-36
P23-37
P23-38
P23-39
P23-40
SRAM_ADD_A218
SRAM_ADD_A217
SRAM_ADD_A216
SRAM_ADD_A215
SRAM_ADD_A214
SRAM_ADD_A213
SRAM_ADD_A212
SRAM_ADD_A211
SRAM_ADD_A210
SRAM_ADD_A209
SRAM_ADD_A208
SRAM_ADD_A207
SRAM_ADD_A206
SRAM_ADD_A205
SRAM_ADD_A204
SRAM_ADD_A203
SRAM_ADD_A202
SRAM_ADD_A201
SRAM_ADD_A200
SRAM_ADD_B218
SRAM_ADD_B217
SRAM_ADD_B216
SRAM_ADD_B215
SRAM_ADD_B214
SRAM_ADD_B213
SRAM_ADD_B212
SRAM_ADD_B211
+3.3V
P23-41
P23-42
P23-43
P23-44
P23-45
P23-46
P23-47
P23-48
P23-49
P23-50
P23-51
P23-52
P23-53
P23-54
P23-55
P23-56
P23-57
P23-58
P23-59
P23-60
P23-61
P23-62
P23-63
P23-64
P23-65
P23-66
P23-67
P23-68
P23-69
P23-70
P23-71
P23-72
P23-73
P23-74
P23-75
P23-76
P23-77
P23-78
P23-79
P23-80
SRAM_ADD_B210
SRAM_ADD_B209
SRAM_ADD_B208
SRAM_ADD_B207
SRAM_ADD_B206
SRAM_ADD_B205
SRAM_ADD_B204
SRAM_ADD_B203
SRAM_ADD_B202
SRAM_ADD_B201
SRAM_ADD_B200
SRAM_DATA26
SRAM_DATA25
SRAM_DATA24
SRAM_DATA23
SRAM_DATA22
SRAM_DATA21
SRAM_DATA20
[5,16,17,18]
DOUT_LAT_2
[5,16,17,18]
/WE_2
[5,16,17,18]
PHASE_B_2
[5,16,17,18]
PHASE_A_2
DAC22_IN6
DAC22_IN5
DAC22_IN4
DAC22_IN3
DAC22_IN2
DAC22_IN1
DAC22_IN0
DAC22_IN[0:6]
[9]
BOARD_SEL23
(2)
P24-1
P24-2
P24-3
P24-4
P24-5
P24-6
P24-7
P24-8
P24-9
P24-10
P24-11
P24-12
P24-13
P24-14
P24-15
P24-16
P24-17
P24-18
P24-19
P24-20
P24-21
P24-22
P24-23
P24-24
P24-25
P24-26
P24-27
P24-28
P24-29
P24-30
P24-31
P24-32
P24-33
P24-34
P24-35
P24-36
P24-37
P24-38
P24-39
P24-40
SRAM_ADD_A218
SRAM_ADD_A217
SRAM_ADD_A216
SRAM_ADD_A215
SRAM_ADD_A214
SRAM_ADD_A213
SRAM_ADD_A212
SRAM_ADD_A211
SRAM_ADD_A210
SRAM_ADD_A209
SRAM_ADD_A208
SRAM_ADD_A207
SRAM_ADD_A206
SRAM_ADD_A205
SRAM_ADD_A204
SRAM_ADD_A203
SRAM_ADD_A202
SRAM_ADD_A201
SRAM_ADD_A200
SRAM_ADD_B218
SRAM_ADD_B217
SRAM_ADD_B216
SRAM_ADD_B215
SRAM_ADD_B214
SRAM_ADD_B213
SRAM_ADD_B212
SRAM_ADD_B211
+3.3V
P24-41
P24-42
P24-43
P24-44
P24-45
P24-46
P24-47
P24-48
P24-49
P24-50
P24-51
P24-52
P24-53
P24-54
P24-55
P24-56
P24-57
P24-58
P24-59
P24-60
P24-61
P24-62
P24-63
P24-64
P24-65
P24-66
P24-67
P24-68
P24-69
P24-70
P24-71
P24-72
P24-73
P24-74
P24-75
P24-76
P24-77
P24-78
P24-79
P24-80
SRAM_ADD_B210
SRAM_ADD_B209
SRAM_ADD_B208
SRAM_ADD_B207
SRAM_ADD_B206
SRAM_ADD_B205
SRAM_ADD_B204
SRAM_ADD_B203
SRAM_ADD_B202
SRAM_ADD_B201
SRAM_ADD_B200
SRAM_DATA26
SRAM_DATA25
SRAM_DATA24
SRAM_DATA23
SRAM_DATA22
SRAM_DATA21
SRAM_DATA20
[5,16,17,18]
DOUT_LAT_2
[5,16,17,18]
/WE_2
[5,16,17,18]
PHASE_B_2
[5,16,17,18]
PHASE_A_2
DAC23_IN6
DAC23_IN5
DAC23_IN4
DAC23_IN3
DAC23_IN2
DAC23_IN1
DAC23_IN0
DAC23_IN[0:6]
[9]
SRAM_ADD_A2[00:18]
[5,16,17,18]
SRAM_ADD_B2[00:18]
[5,16,17,18]
SRAM_DATA2[0:6]
[5,16,17,18]
B
ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203)
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE
4/12/00

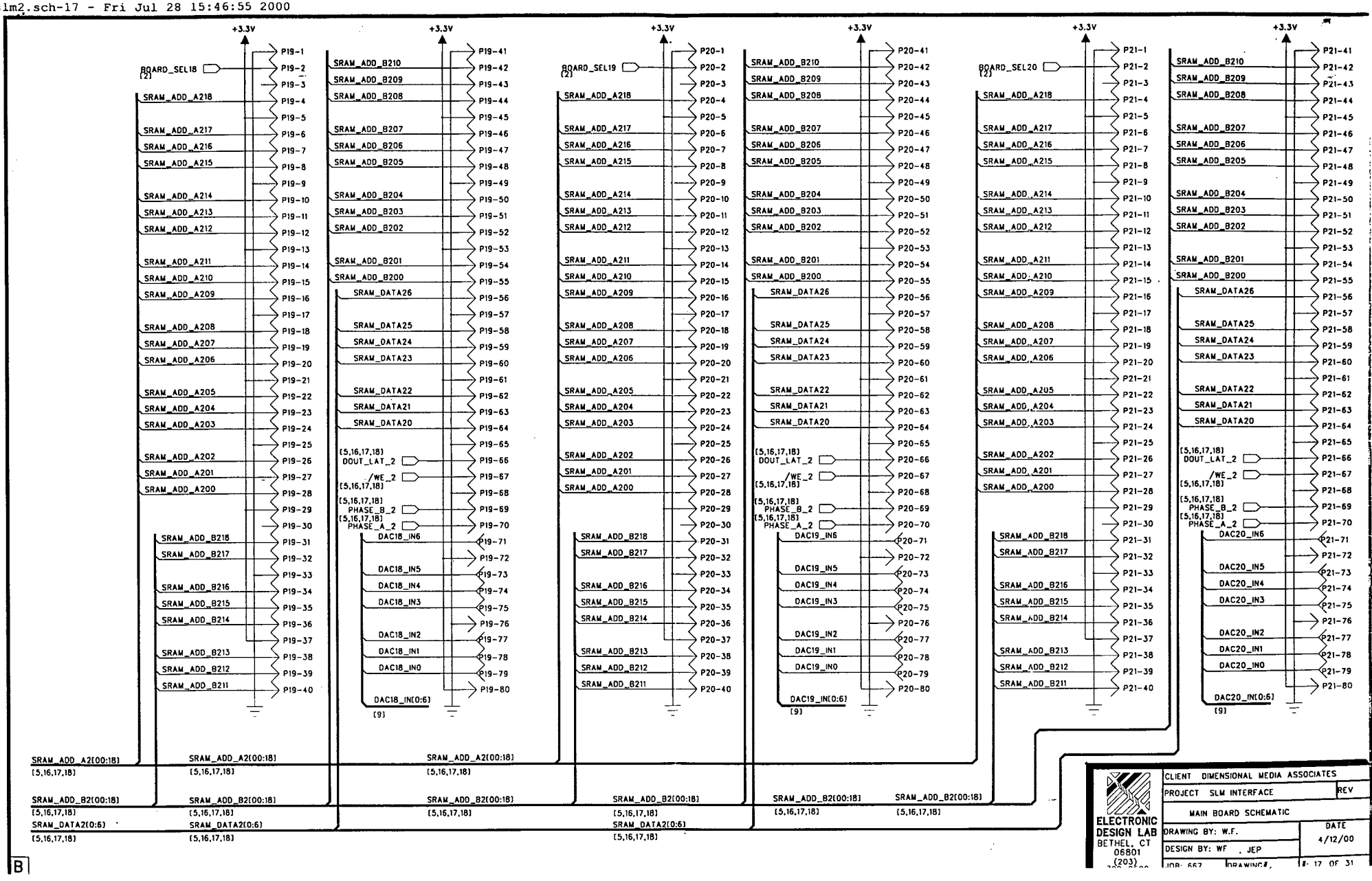

+3.3V
BOARD_SEL24
(2)
BOARD_SEL25
(2)
BOARD_SEL26
(2)
SRAM_ADD_A318
SRAM_ADD_A317
SRAM_ADD_A316
SRAM_ADD_A315
SRAM_ADD_A314
SRAM_ADD_A313
SRAM_ADD_A312
SRAM_ADD_A311
SRAM_ADD_A310
SRAM_ADD_A309
SRAM_ADD_A308
SRAM_ADD_A307
SRAM_ADD_A306
SRAM_ADD_A305
SRAM_ADD_A304
SRAM_ADD_A303
SRAM_ADD_A302
SRAM_ADD_A301
SRAM_ADD_A300
SRAM_ADD_B318
SRAM_ADD_B317
SRAM_ADD_B316
SRAM_ADD_B315
SRAM_ADD_B314
SRAM_ADD_B313
SRAM_ADD_B312
SRAM_ADD_B311
SRAM_ADD_B310
SRAM_ADD_B309
SRAM_ADD_B308
SRAM_ADD_B307
SRAM_ADD_B306
SRAM_ADD_B305
SRAM_ADD_B304
SRAM_ADD_B303
SRAM_ADD_B302
SRAM_ADD_B301
SRAM_ADD_B300
SRAM_DATA36
SRAM_DATA35
SRAM_DATA34
SRAM_DATA33
SRAM_DATA32
SRAM_DATA31
SRAM_DATA30
(6,19,20,21) DOUT_LAT_3
/WE_3 (6,19,20,21)
(6,19,20,21) PHASE_B_3
(6,19,20,21) PHASE_A_3
DAC24_IN6
DAC24_IN5
DAC24_IN4
DAC24_IN3
DAC24_IN2
DAC24_IN1
DAC24_IN0
DAC24_IN[0:6]
(10)
DAC25_IN6
DAC25_IN5
DAC25_IN4
DAC25_IN3
DAC25_IN2
DAC25_IN1
DAC25_IN0
DAC25_IN[0:6]
(10)
DAC26_IN6
DAC26_IN5
DAC26_IN4
DAC26_IN3
DAC26_IN2
DAC26_IN1
DAC26_IN0
DAC26_IN[0:6]
(10)
SRAM_ADD_A3[00:18]
(6,19,20,21)
SRAM_ADD_B3[00:18]
(6,19,20,21)
SRAM_DATA3[0:6]
(6,19,20,21)
B
ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203)
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE 4/12/00

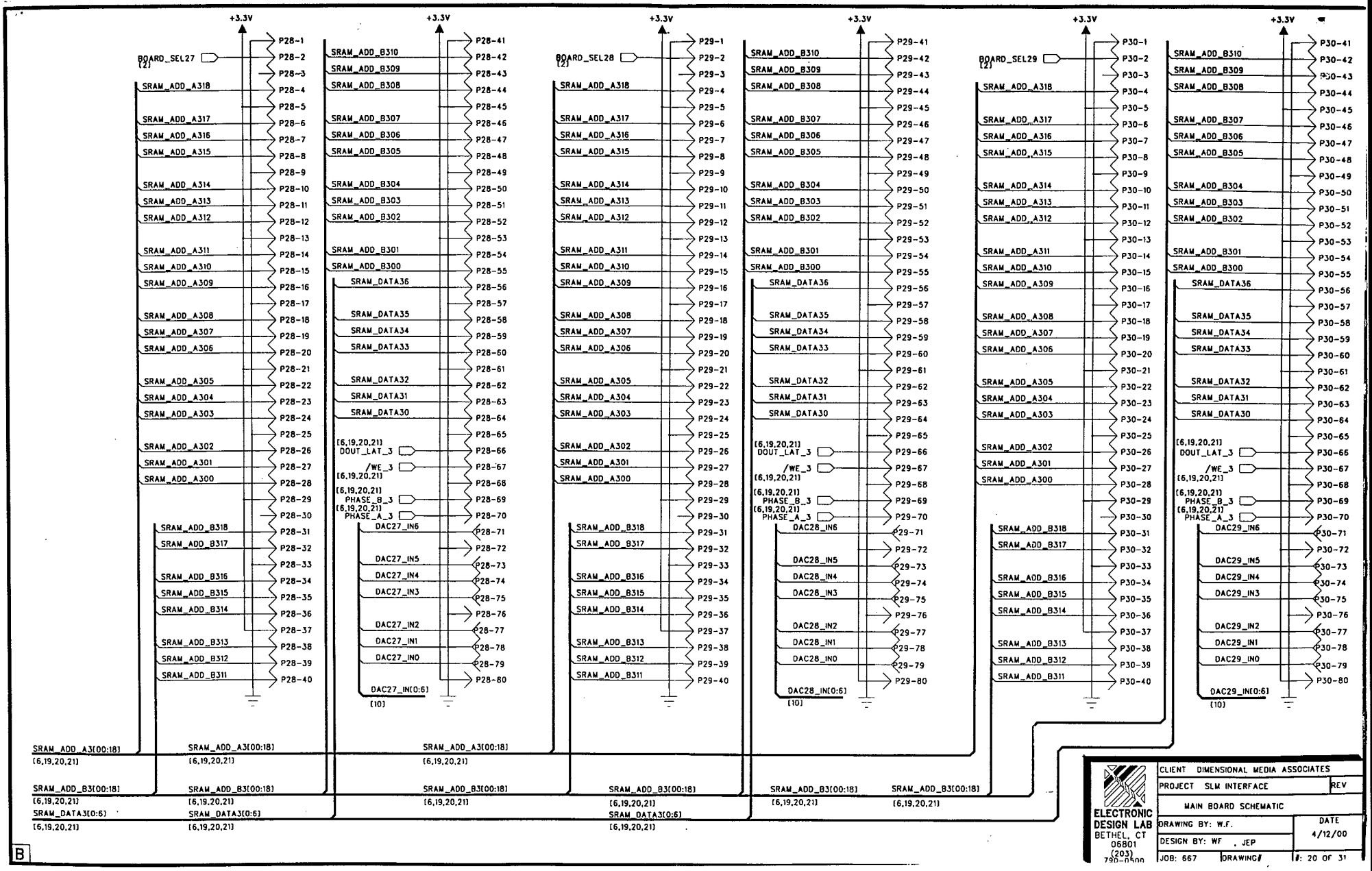


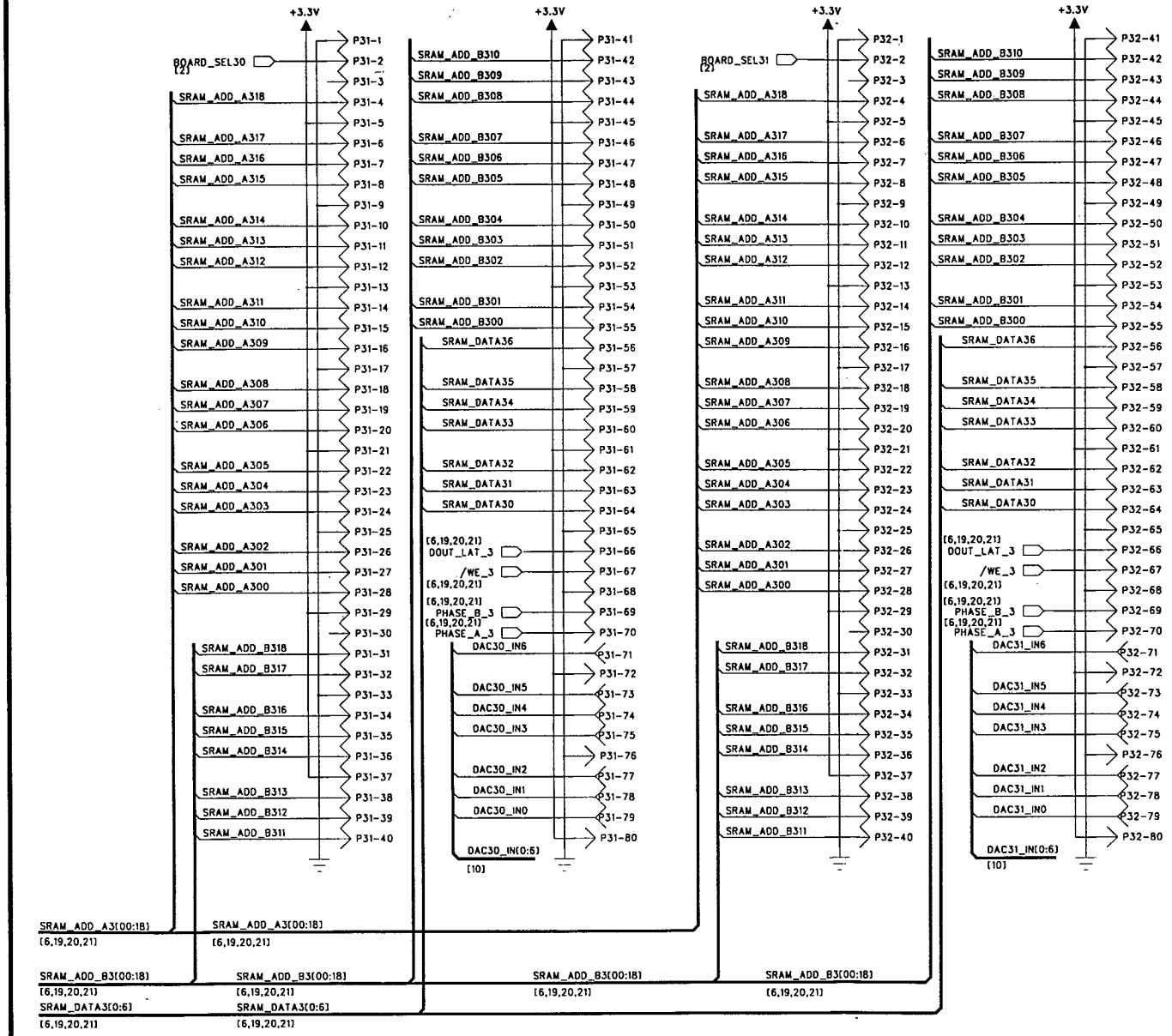
ELECTRONIC DESIGN LAB, BETHEL, CT 06801, (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING# — # 15 OF 31



ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203)	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#:	# 16 OF 31

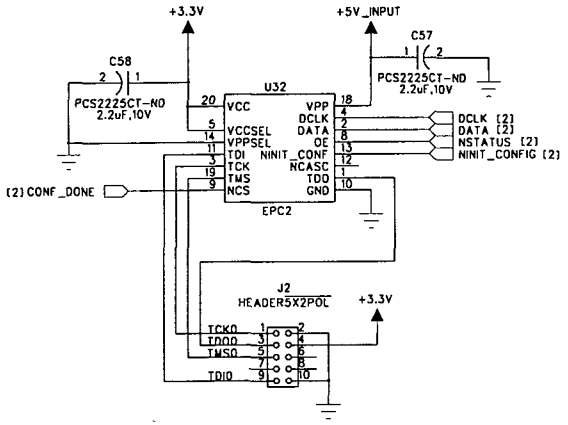

+3.3V
BOARD_SEL18
BOARD_SEL19
BOARD_SEL20
SRAM_ADD_A218
SRAM_ADD_A217
SRAM_ADD_A216
SRAM_ADD_A215
SRAM_ADD_A214
SRAM_ADD_A213
SRAM_ADD_A212
SRAM_ADD_A211
SRAM_ADD_A210
SRAM_ADD_A209
SRAM_ADD_A208
SRAM_ADD_A207
SRAM_ADD_A206
SRAM_ADD_A205
SRAM_ADD_A204
SRAM_ADD_A203
SRAM_ADD_A202
SRAM_ADD_A201
SRAM_ADD_A200
SRAM_ADD_B218
SRAM_ADD_B217
SRAM_ADD_B216
SRAM_ADD_B215
SRAM_ADD_B214
SRAM_ADD_B213
SRAM_ADD_B212
SRAM_ADD_B211
SRAM_ADD_B210
SRAM_ADD_B209
SRAM_ADD_B208
SRAM_ADD_B207
SRAM_ADD_B206
SRAM_ADD_B205
SRAM_ADD_B204
SRAM_ADD_B203
SRAM_ADD_B202
SRAM_ADD_B201
SRAM_ADD_B200
SRAM_DATA26
SRAM_DATA25
SRAM_DATA24
SRAM_DATA23
SRAM_DATA22
SRAM_DATA21
SRAM_DATA20
DOUT_LAT_2
/WE_2
PHASE_B_2
PHASE_A_2
DAC18_IN6
DAC18_IN5
DAC18_IN4
DAC18_IN3
DAC18_IN2
DAC18_IN1
DAC18_IN0
DAC18_IN[0:6]
DAC19_IN6
DAC19_IN5
DAC19_IN4
DAC19_IN3
DAC19_IN2
DAC19_IN1
DAC19_IN0
DAC19_IN[0:6]
DAC20_IN6
DAC20_IN5
DAC20_IN4
DAC20_IN3
DAC20_IN2
DAC20_IN1
DAC20_IN0
DAC20_IN[0:6]
SRAM_ADD_A2[00:18]
SRAM_ADD_B2[00:18]
SRAM_DATA2[0:6]
[5,16,17,18]
[9]
(2)
B
ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203)
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DATE 4/12/00
DESIGN BY: WF , JEP
JOB: 667
DRAWING#:
17 OF 31


+3.3V
BOARD_SEL27
[2]
SRAM_ADD_A318
SRAM_ADD_A317
SRAM_ADD_A316
SRAM_ADD_A315
SRAM_ADD_A314
SRAM_ADD_A313
SRAM_ADD_A312
SRAM_ADD_A311
SRAM_ADD_A310
SRAM_ADD_A309
SRAM_ADD_A308
SRAM_ADD_A307
SRAM_ADD_A306
SRAM_ADD_A305
SRAM_ADD_A304
SRAM_ADD_A303
SRAM_ADD_A302
SRAM_ADD_A301
SRAM_ADD_A300
SRAM_ADD_B318
SRAM_ADD_B317
SRAM_ADD_B316
SRAM_ADD_B315
SRAM_ADD_B314
SRAM_ADD_B313
SRAM_ADD_B312
SRAM_ADD_B311
P28-1
P28-2
P28-3
P28-4
P28-5
P28-6
P28-7
P28-8
P28-9
P28-10
P28-11
P28-12
P28-13
P28-14
P28-15
P28-16
P28-17
P28-18
P28-19
P28-20
P28-21
P28-22
P28-23
P28-24
P28-25
P28-26
P28-27
P28-28
P28-29
P28-30
P28-31
P28-32
P28-33
P28-34
P28-35
P28-36
P28-37
P28-38
P28-39
P28-40
SRAM_ADD_B310
SRAM_ADD_B309
SRAM_ADD_B308
SRAM_ADD_B307
SRAM_ADD_B306
SRAM_ADD_B305
SRAM_ADD_B304
SRAM_ADD_B303
SRAM_ADD_B302
SRAM_ADD_B301
SRAM_ADD_B300
SRAM_DATA36
SRAM_DATA35
SRAM_DATA34
SRAM_DATA33
SRAM_DATA32
SRAM_DATA31
SRAM_DATA30
[6,19,20,21]
DOUT_LAT_3
/WE_3
[6,19,20,21]
[6,19,20,21]
PHASE_B_3
[6,19,20,21]
PHASE_A_3
DAC27_IN6
DAC27_IN5
DAC27_IN4
DAC27_IN3
DAC27_IN2
DAC27_IN1
DAC27_IN0
DAC27_IN[0:6]
[10]
P28-41
P28-42
P28-43
P28-44
P28-45
P28-46
P28-47
P28-48
P28-49
P28-50
P28-51
P28-52
P28-53
P28-54
P28-55
P28-56
P28-57
P28-58
P28-59
P28-60
P28-61
P28-62
P28-63
P28-64
P28-65
P28-66
P28-67
P28-68
P28-69
P28-70
P28-71
P28-72
P28-73
P28-74
P28-75
P28-76
P28-77
P28-78
P28-79
P28-80
BOARD_SEL28
[2]
P29-1
P29-2
P29-3
P29-4
P29-5
P29-6
P29-7
P29-8
P29-9
P29-10
P29-11
P29-12
P29-13
P29-14
P29-15
P29-16
P29-17
P29-18
P29-19
P29-20
P29-21
P29-22
P29-23
P29-24
P29-25
P29-26
P29-27
P29-28
P29-29
P29-30
P29-31
P29-32
P29-33
P29-34
P29-35
P29-36
P29-37
P29-38
P29-39
P29-40
P29-41
P29-42
P29-43
P29-44
P29-45
P29-46
P29-47
P29-48
P29-49
P29-50
P29-51
P29-52
P29-53
P29-54
P29-55
P29-56
P29-57
P29-58
P29-59
P29-60
P29-61
P29-62
P29-63
P29-64
P29-65
P29-66
P29-67
P29-68
P29-69
P29-70
P29-71
P29-72
P29-73
P29-74
P29-75
P29-76
P29-77
P29-78
P29-79
P29-80
DAC28_IN6
DAC28_IN5
DAC28_IN4
DAC28_IN3
DAC28_IN2
DAC28_IN1
DAC28_IN0
DAC28_IN[0:6]
[10]
BOARD_SEL29
[2]
P30-1
P30-2
P30-3
P30-4
P30-5
P30-6
P30-7
P30-8
P30-9
P30-10
P30-11
P30-12
P30-13
P30-14
P30-15
P30-16
P30-17
P30-18
P30-19
P30-20
P30-21
P30-22
P30-23
P30-24
P30-25
P30-26
P30-27
P30-28
P30-29
P30-30
P30-31
P30-32
P30-33
P30-34
P30-35
P30-36
P30-37
P30-38
P30-39
P30-40
P30-41
P30-42
P30-43
P30-44
P30-45
P30-46
P30-47
P30-48
P30-49
P30-50
P30-51
P30-52
P30-53
P30-54
P30-55
P30-56
P30-57
P30-58
P30-59
P30-60
P30-61
P30-62
P30-63
P30-64
P30-65
P30-66
P30-67
P30-68
P30-69
P30-70
P30-71
P30-72
P30-73
P30-74
P30-75
P30-76
P30-77
P30-78
P30-79
P30-80
DAC29_IN6
DAC29_IN5
DAC29_IN4
DAC29_IN3
DAC29_IN2
DAC29_IN1
DAC29_IN0
DAC29_IN[0:6]
[10]
SRAM_ADD_A3[00:18]
[6,19,20,21]
SRAM_ADD_B3[00:18]
[6,19,20,21]
SRAM_DATA3[0:6]
[6,19,20,21]
B
ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500
CLIENT DIMENSIONAL MEDIA ASSOCIATES
PROJECT SLM INTERFACE
REV
MAIN BOARD SCHEMATIC
DRAWING BY: W.F.
DESIGN BY: WF , JEP
DATE
4/12/00
JOB: 667
DRAWING#
#: 20 OF 31

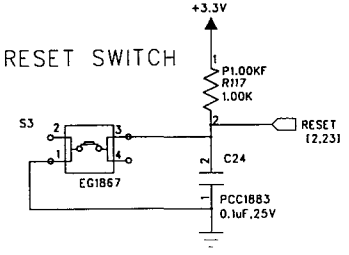


ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#	#: 21 OF 31

FLEX 10K PROGRAM MEMORY

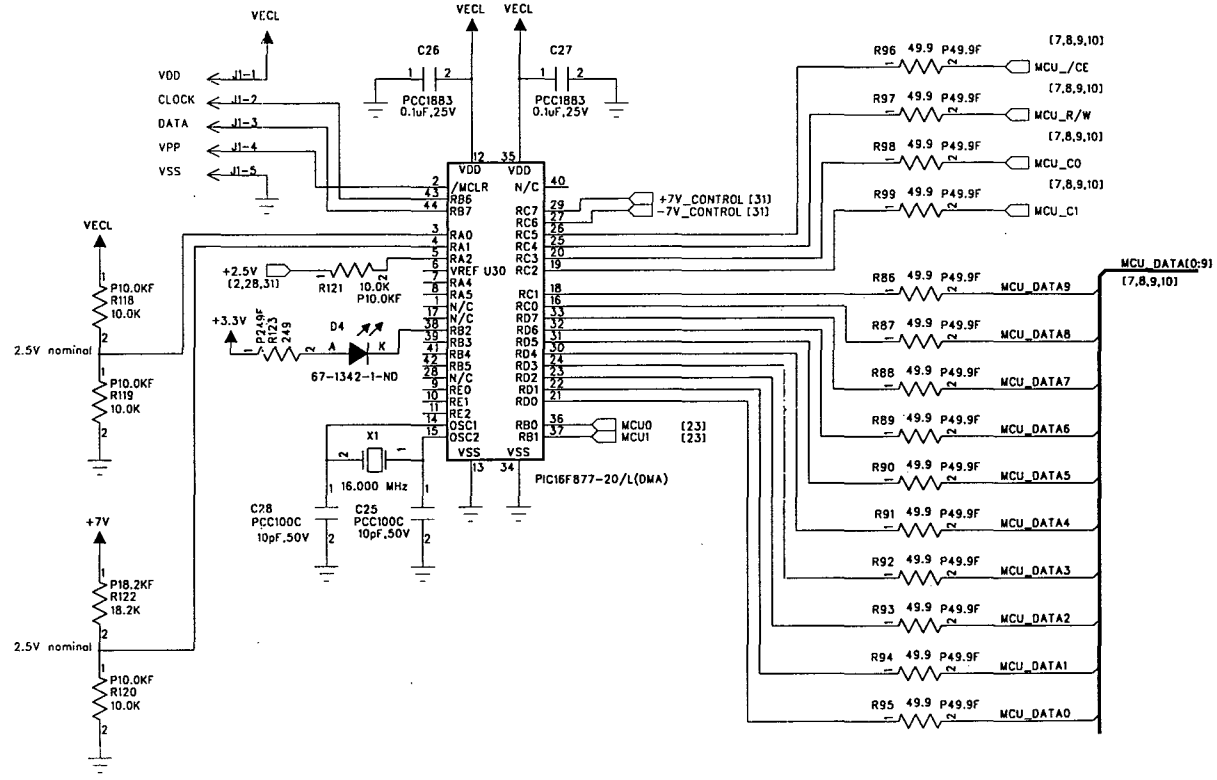


RESET SWITCH



MICROCONTROLLER(MCU)

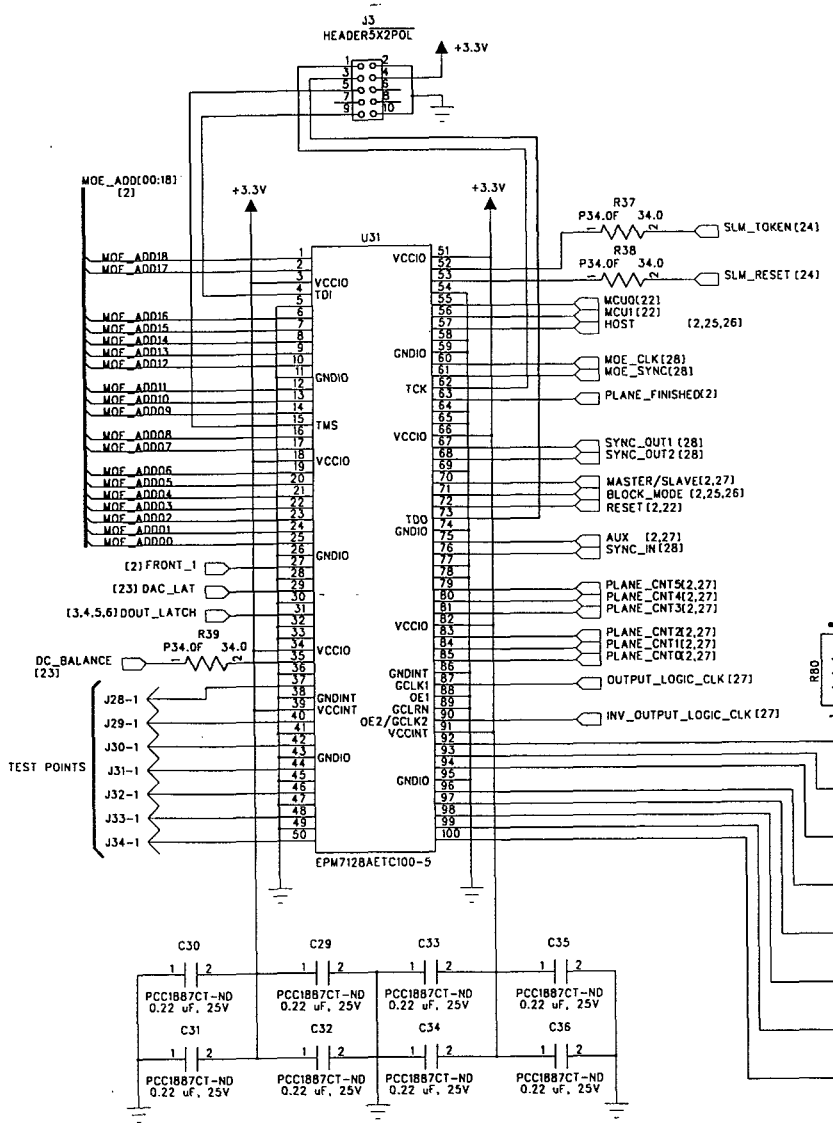
For RAMDAC programming and Power Supply Sequencing.



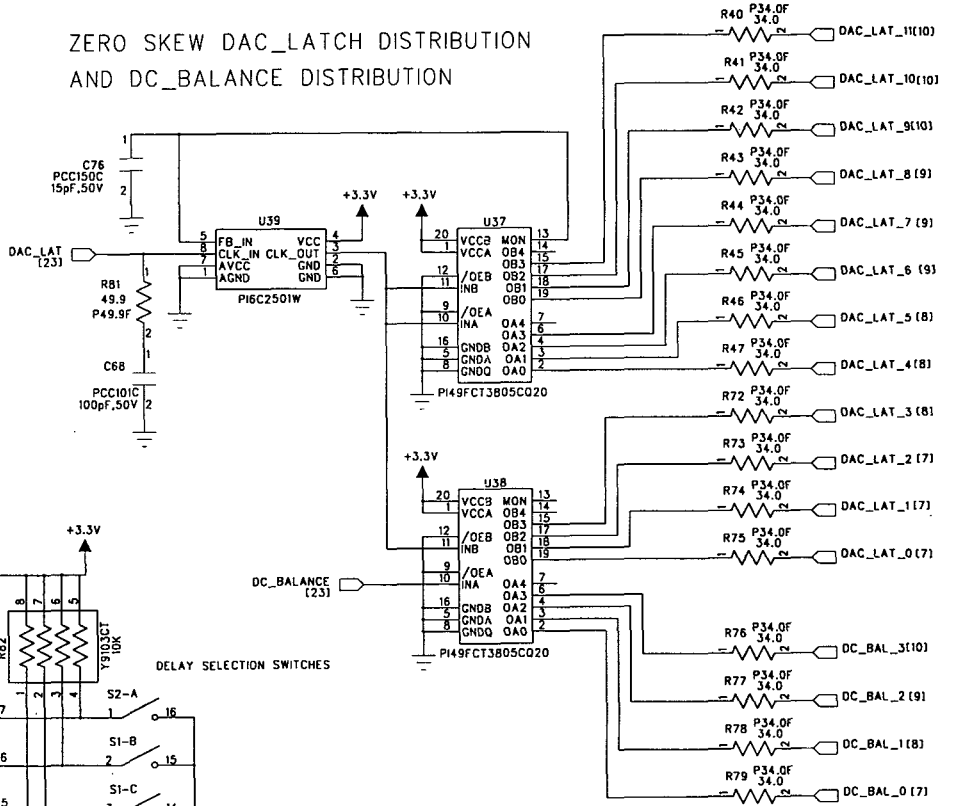
B

ELECTRONIC DESIGN LAB, BETHEL, CT 06801, (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING #: 22 OF 31

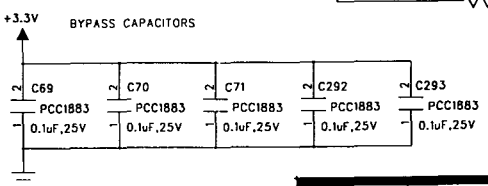
SLM OUTPUT LOGIC



ZERO SKEW DAC_LATCH DISTRIBUTION AND DC_BALANCE DISTRIBUTION



BYPASS CAPACITORS

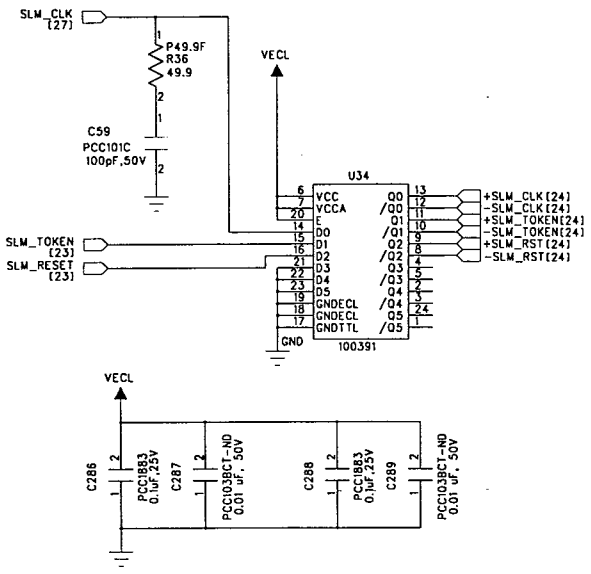


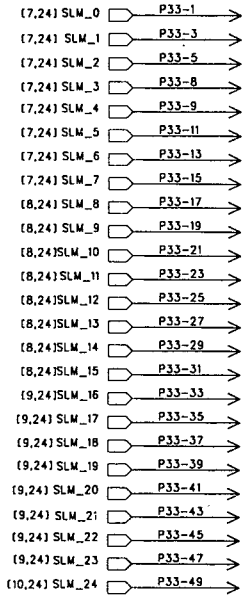
ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES
	PROJECT SLM INTERFACE — REV
	MAIN BOARD SCHEMATIC
	DRAWING BY: W.F. — DATE 4/12/00
	DESIGN BY: WF , JEP
	JOB: 667 — DRAWING #: 23 OF 31

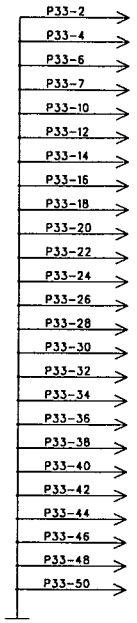
B

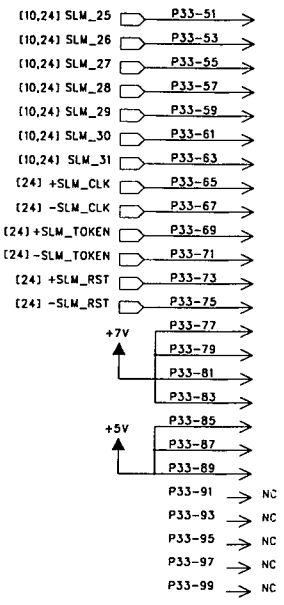
OUTPUT CONNECTOR; TRANSLATOR AND TERMINATIONS

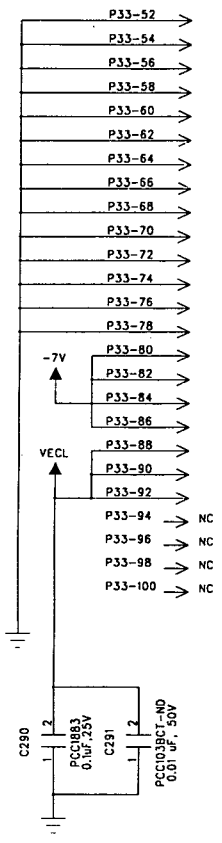
SLM TTL TO PECL TRANSLATOR



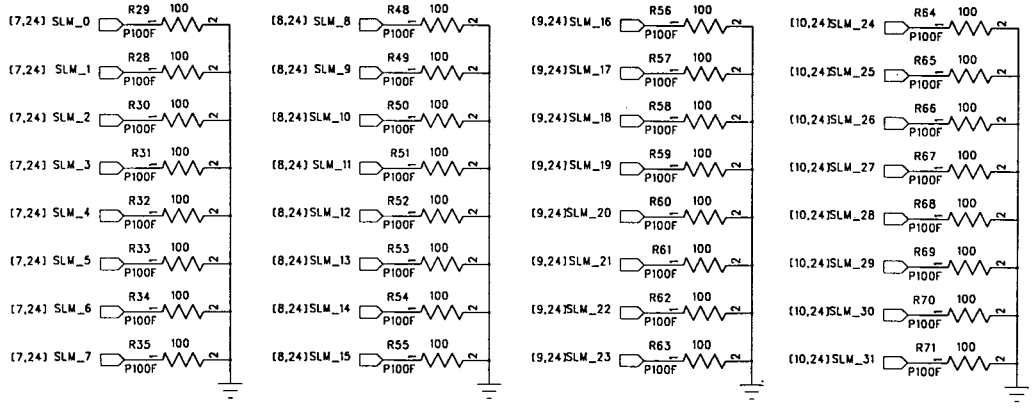








DATA SIGNAL SOURCE END TERMINATION RESISTORS

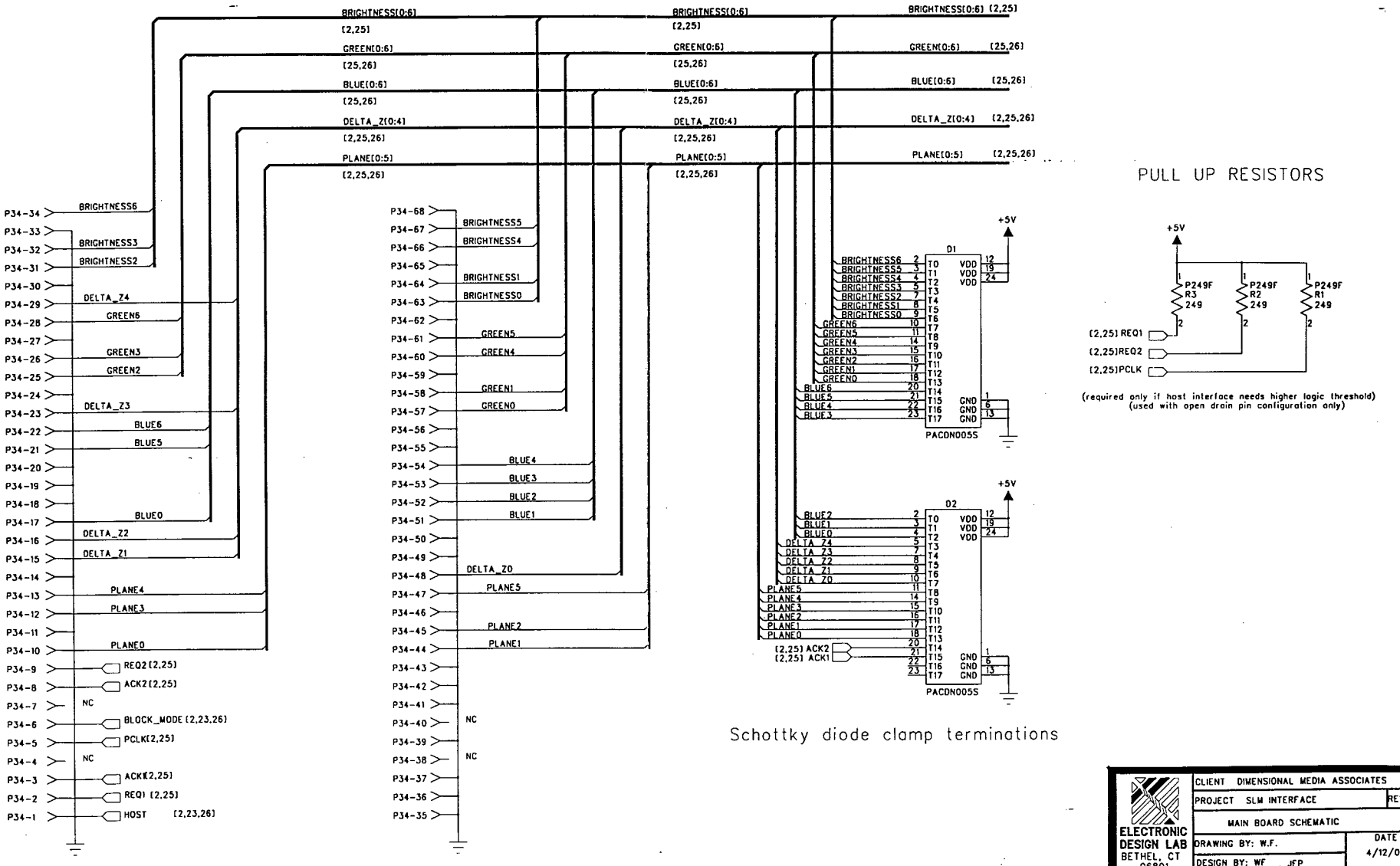


B

ELECTRONIC DESIGN LAB
BETHEL, CT 06801
(203)

CLIENT DIMENSIONAL MEDIA ASSOCIATES	
PROJECT SLM INTERFACE	REV
MAIN BOARD SCHEMATIC	
DRAWING BY: W.F.	DATE 4/12/00
DESIGN BY: WF , JEP	

INPUT CONNECTOR AND TERMINATIONS



Schottky diode clamp terminations

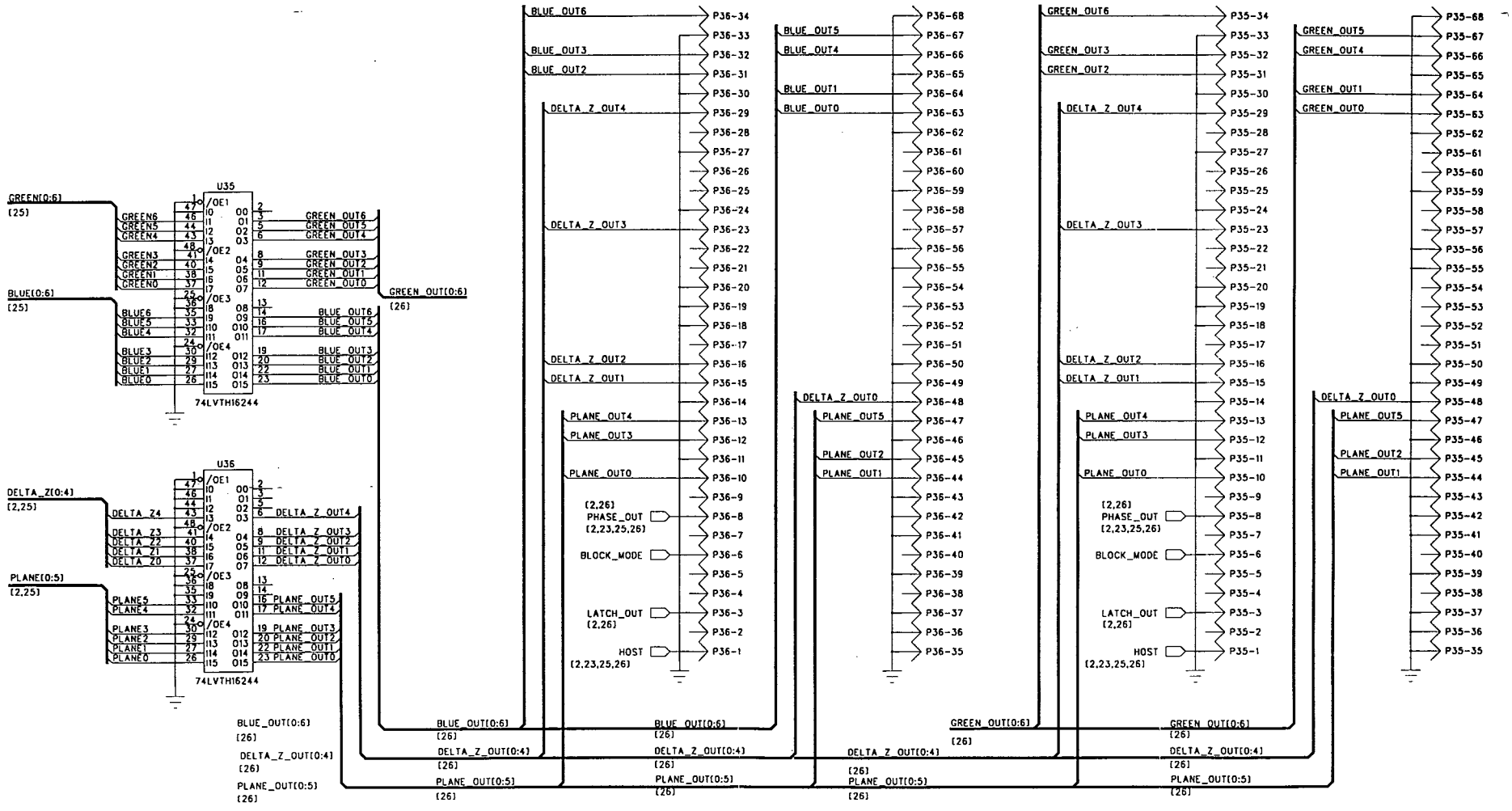
PULL UP RESISTORS

(required only if host interface needs higher logic threshold)
(used with open drain pin configuration only)

ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#	#: 25 OF 31

B

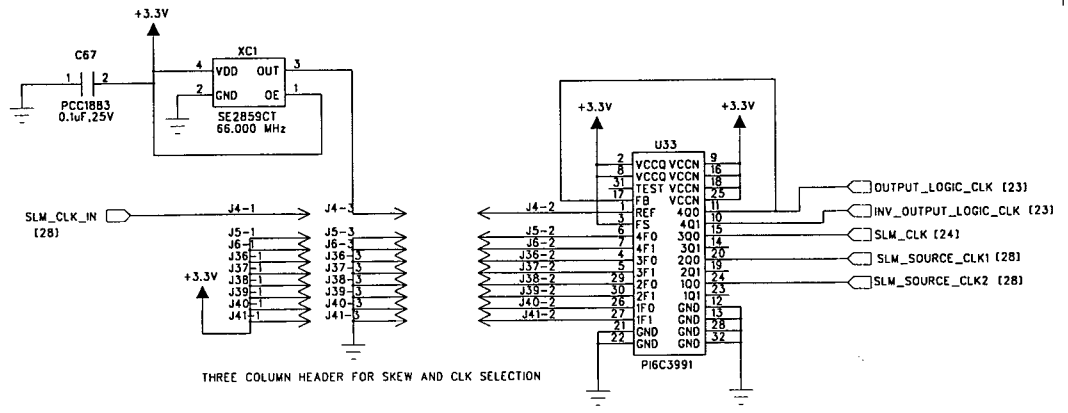
OUTPUTS TO SLAVE CARDS



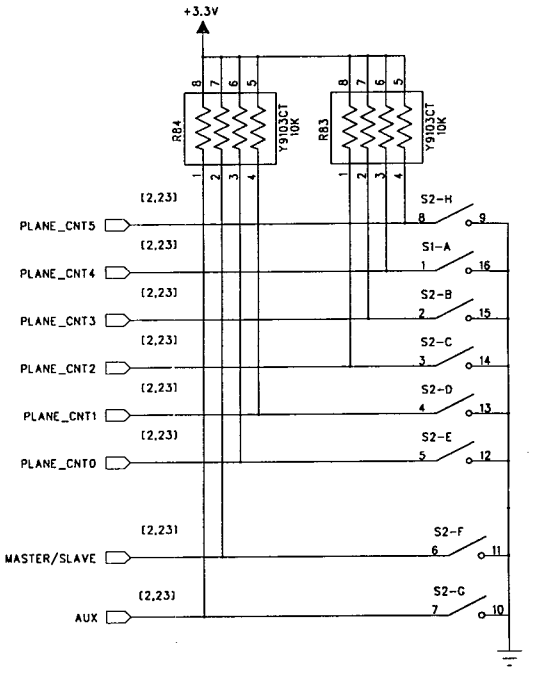
ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#	#: 26 OF 31

B

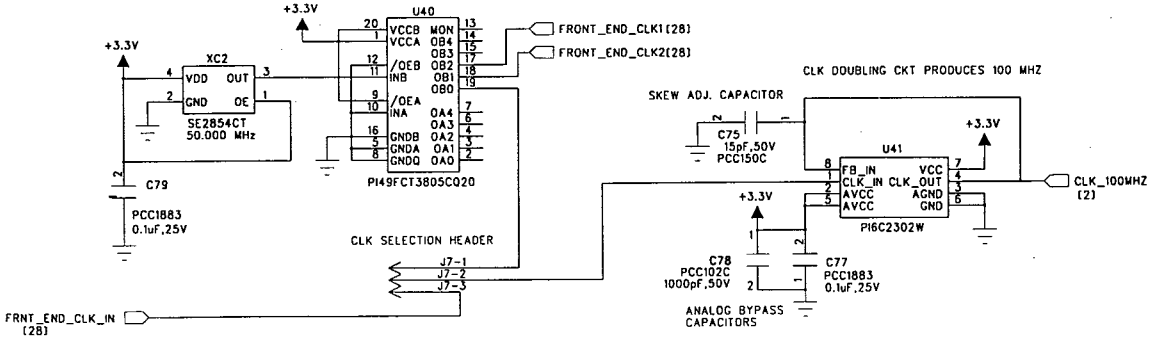
SLM CLOCK SOURCE AND DISTRIBUTION



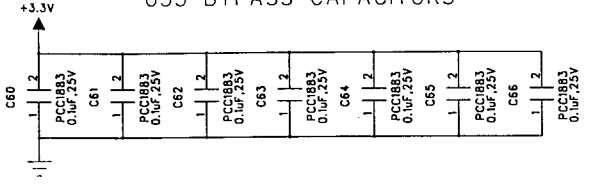
PLANE COUNT, MASTER/SLAVE AND AUX SELECTION SWITCHES



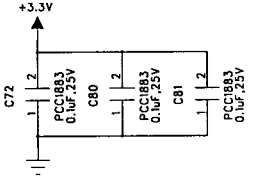
FRONT END CLOCK SOURCE AND DISTRIBUTION



U33 BYPASS CAPACITORS



U40, U41 BYPASS CAPACITORS

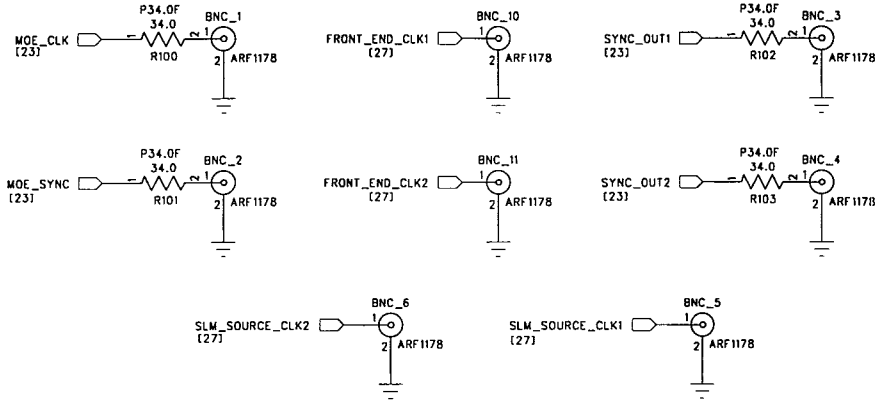


ELECTRONIC DESIGN LAB, BETHEL, CT 06801, (203) 790-0500	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#:	#: 27 OF 31

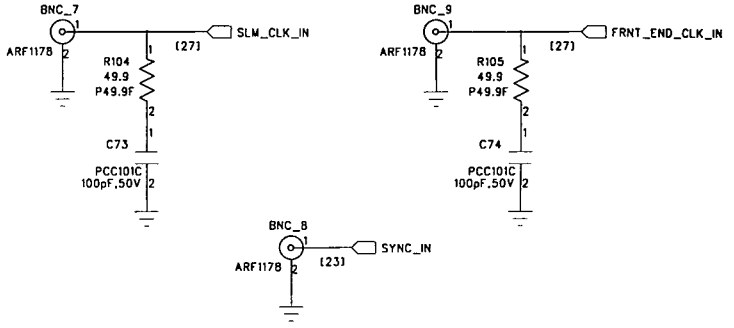
B

BNC CONNECTORS WITH TERMINATIONS

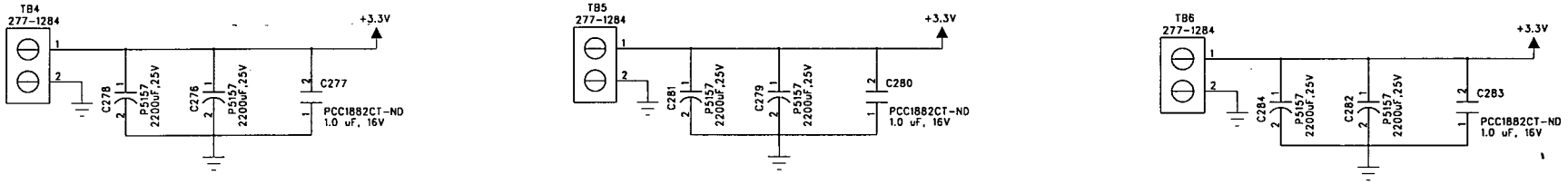
OUTPUTS



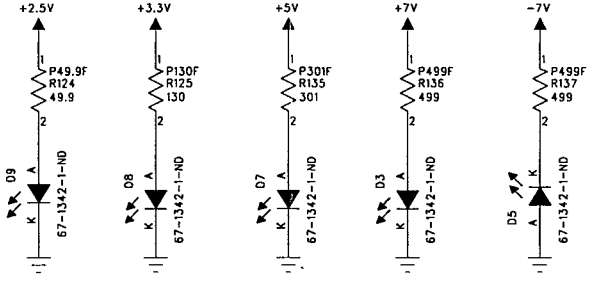
INPUTS



+3.3V POWER CONNECTORS AND FILTER CAPACITORS



POWER INDICATORS

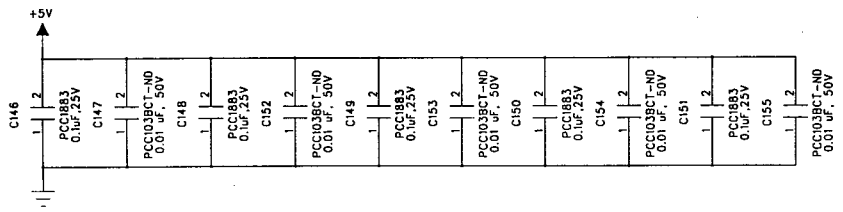


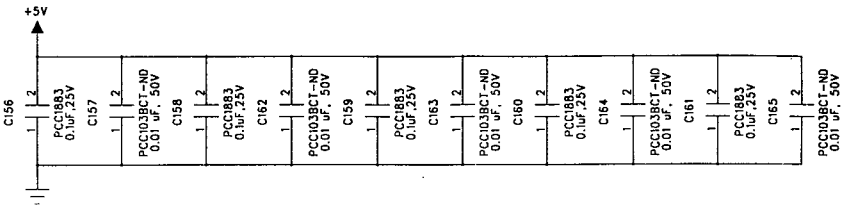
B

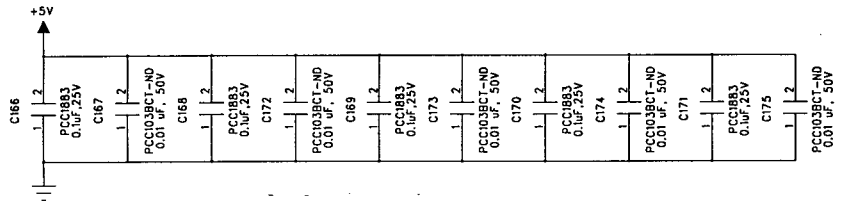
ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500

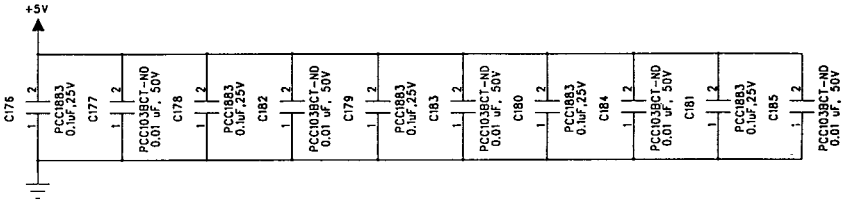
CLIENT DIMENSIONAL MEDIA ASSOCIATES		
PROJECT SLM INTERFACE		REV
MAIN BOARD SCHEMATIC		
DRAWING BY: W.F.		DATE 4/12/00
DESIGN BY: WF , JEP		
JOB: 667	DRAWING#	#: 28 OF 31

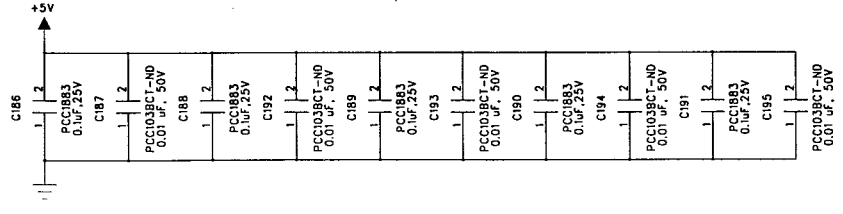
RAMDAC VAA BYPASS CAPACITORS

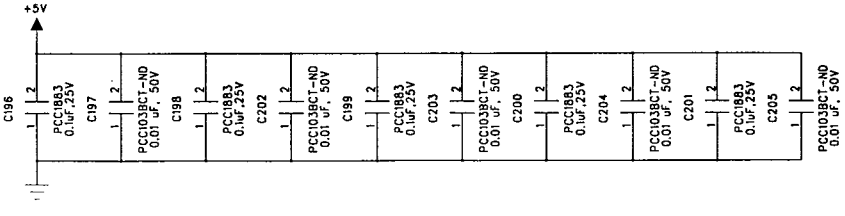










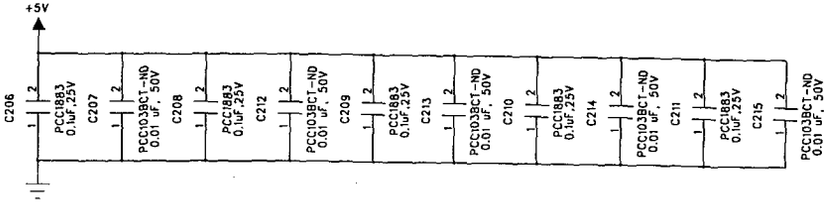


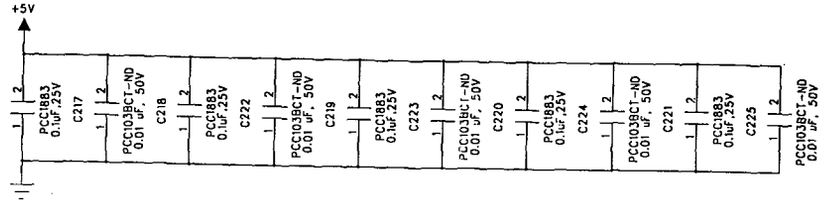
B

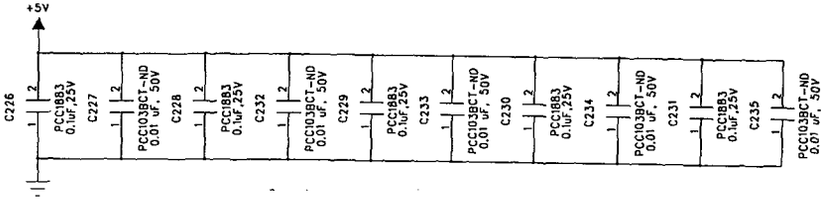
ELECTRONIC DESIGN LAB
BETHEL, CT
06801
(203) 790-0500

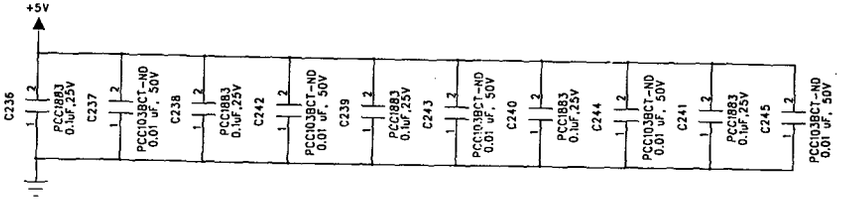
CLIENT DIMENSIONAL MEDIA ASSOCIATES		
PROJECT SLM INTERFACE		REV
MAIN BOARD SCHEMATIC		
DRAWING BY: W.F.		DATE 4/12/00
DESIGN BY: WF , JEP		
JOB: 667	DRAWING#	#: 29 OF 31

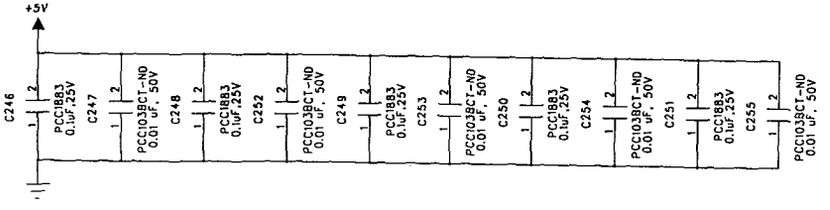
RAMDAC VAA BYPASS CAPACITORS

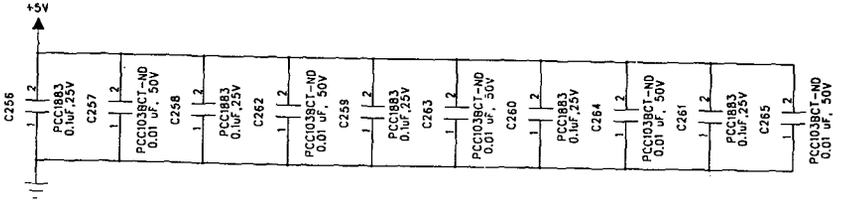








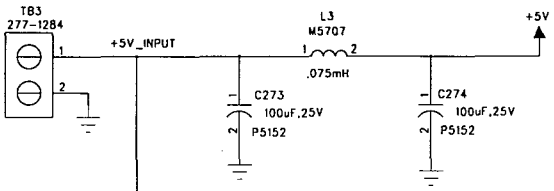




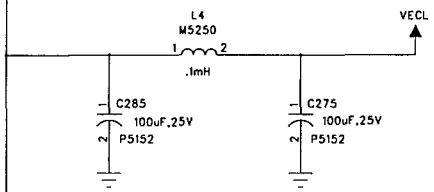
B

ELECTRONIC DESIGN LAB BETHEL, CT 06801	CLIENT DIMENSIONAL MEDIA ASSOCIATES	
	PROJECT SLM INTERFACE	REV
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF . JEP	

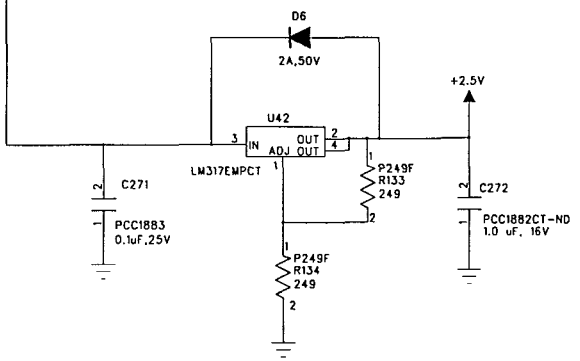
+5V ANALOG SUPPLY VOLTAGE FILTER



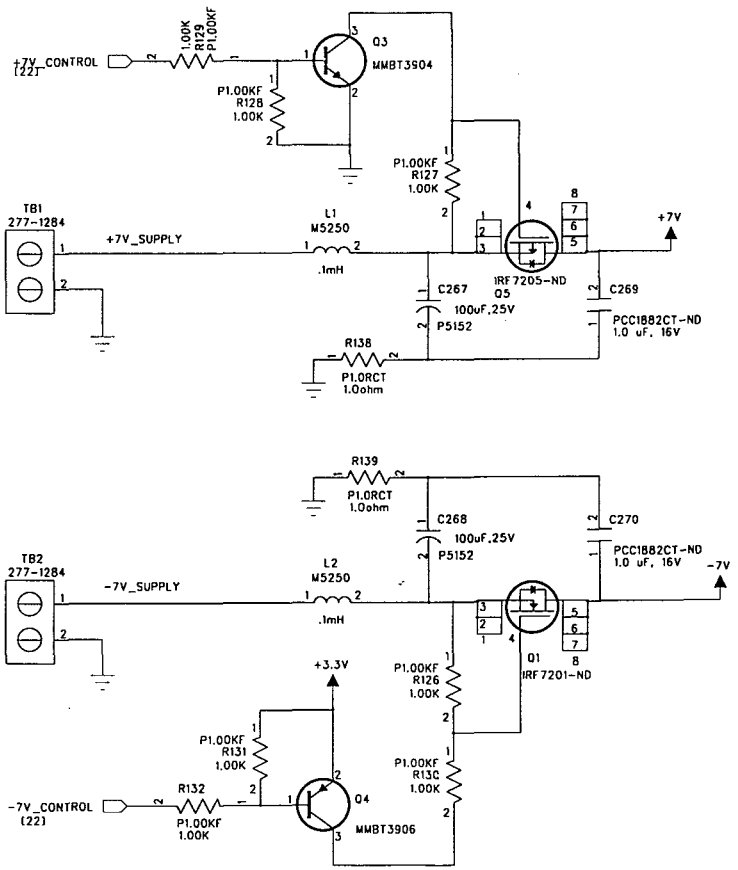
VECL SUPPLY VOLTAGE FILTER



+2.5V SUPPLY REGULATOR

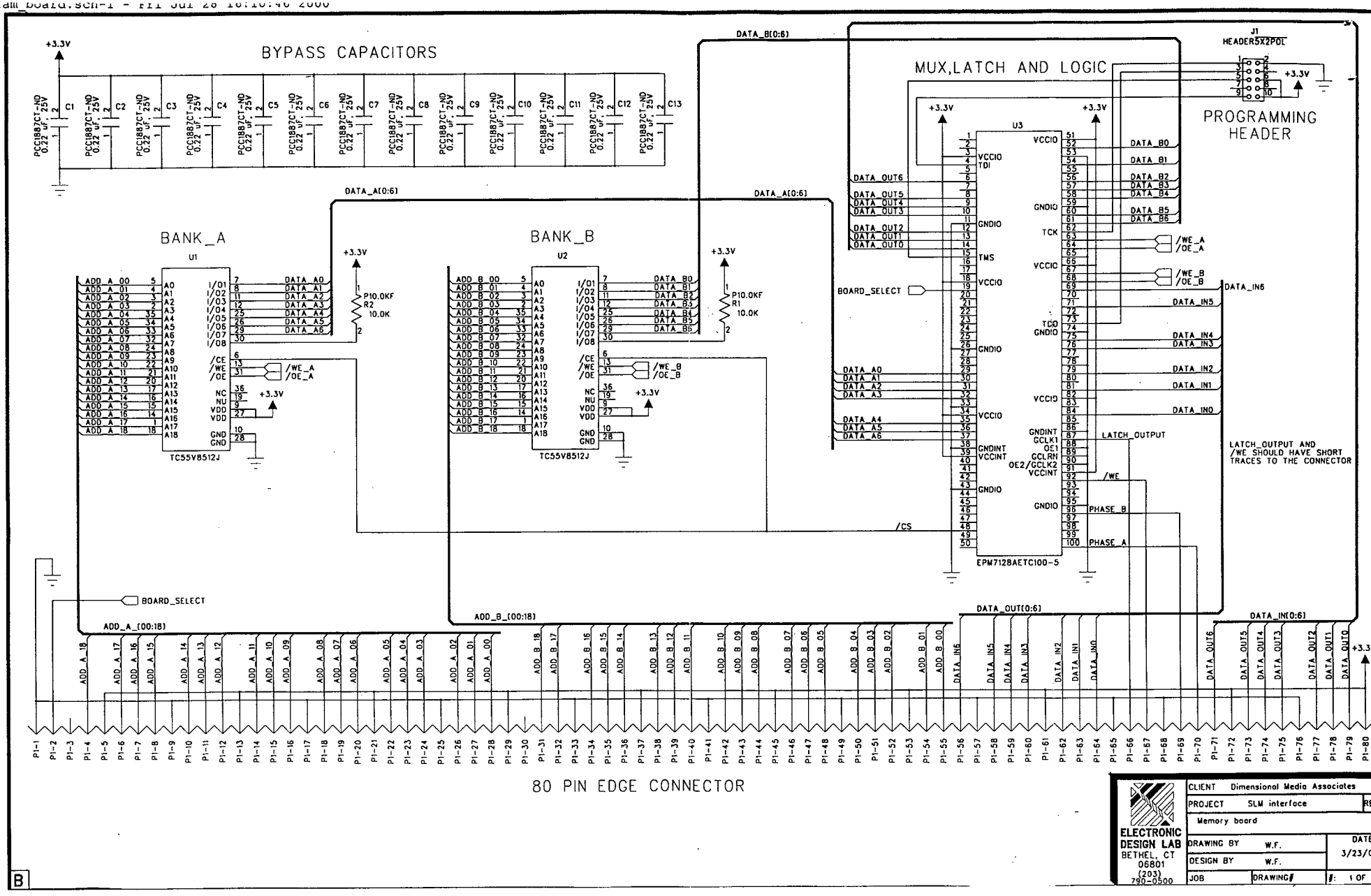


ANALOG SUPPLY VOLTAGE FILTERS AND SWITCHS



ELECTRONIC DESIGN LAB BETHEL, CT 06801 (203)	CLIENT DIMENSIONAL MEDIA ASSOCIATES	REV
	PROJECT SLM INTERFACE	
	MAIN BOARD SCHEMATIC	
	DRAWING BY: W.F.	DATE 4/12/00
	DESIGN BY: WF , JEP	
	JOB: 667 DRAWING#	#: 31 OF 31

B

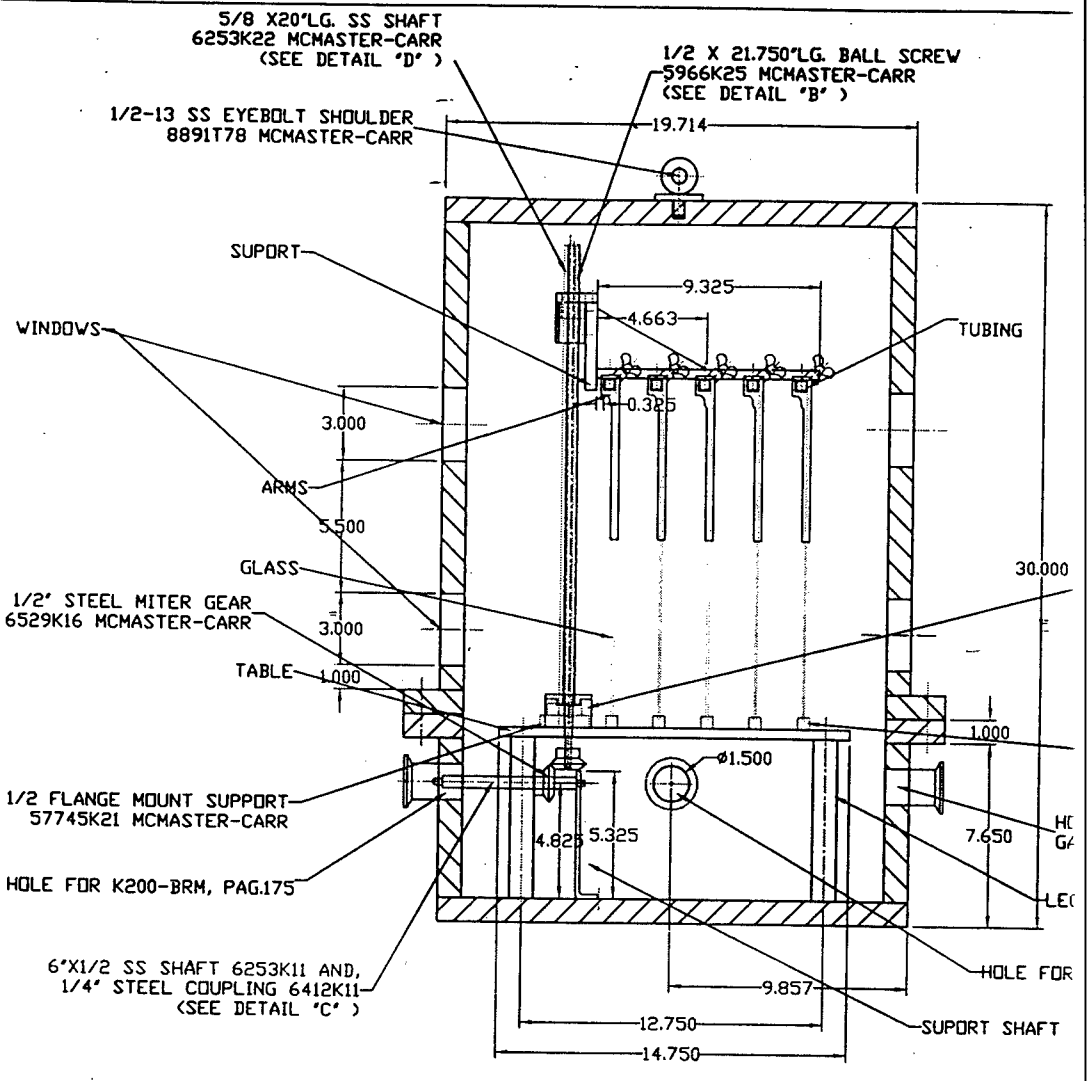


ELECTRONIC DESIGN LAB, BETHEL, CT 06801 (203) 790-0500	CLIENT: Dimensional Media Associates	
	PROJECT: SLM interface	REV
	Memory board	
	DRAWING BY: W.F.	DATE: 3/23/00
	DESIGN BY: W.F.	
	JOB	DRAWING #: 1 OF 1

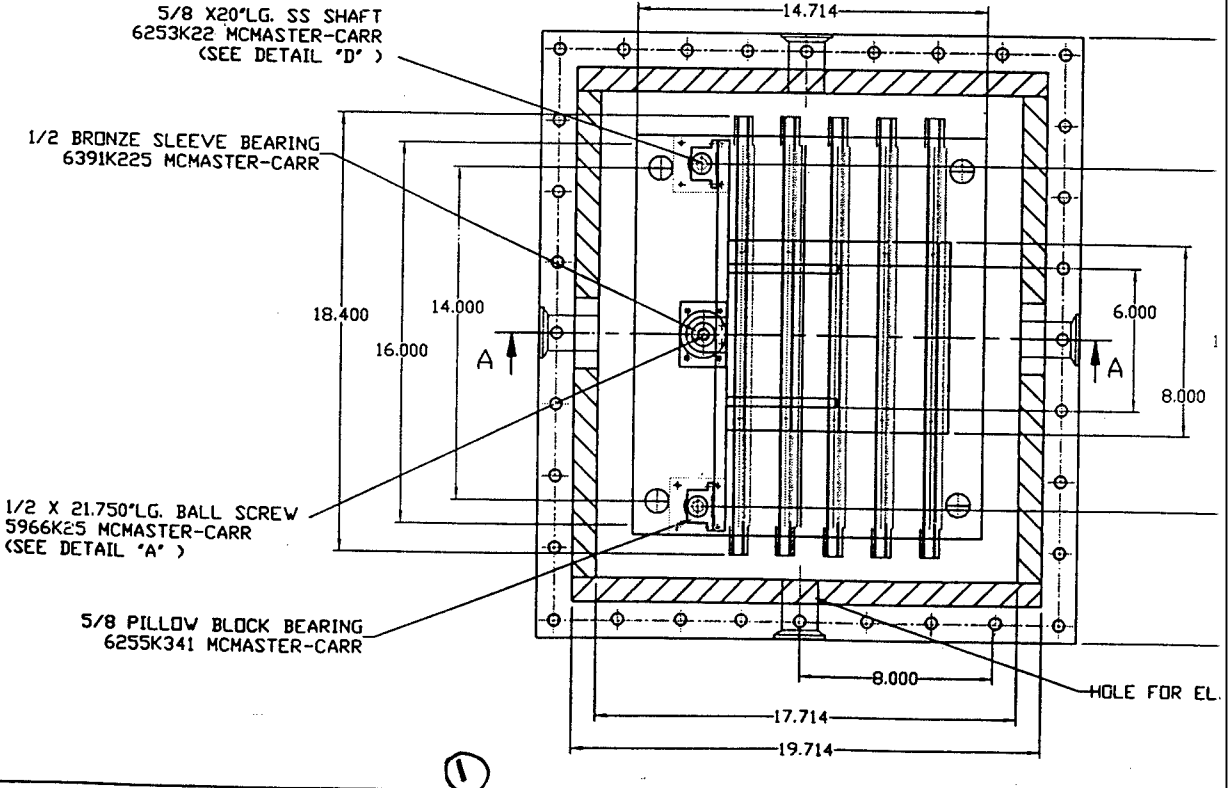
B

F. Vacuum chamber mechanical drawings

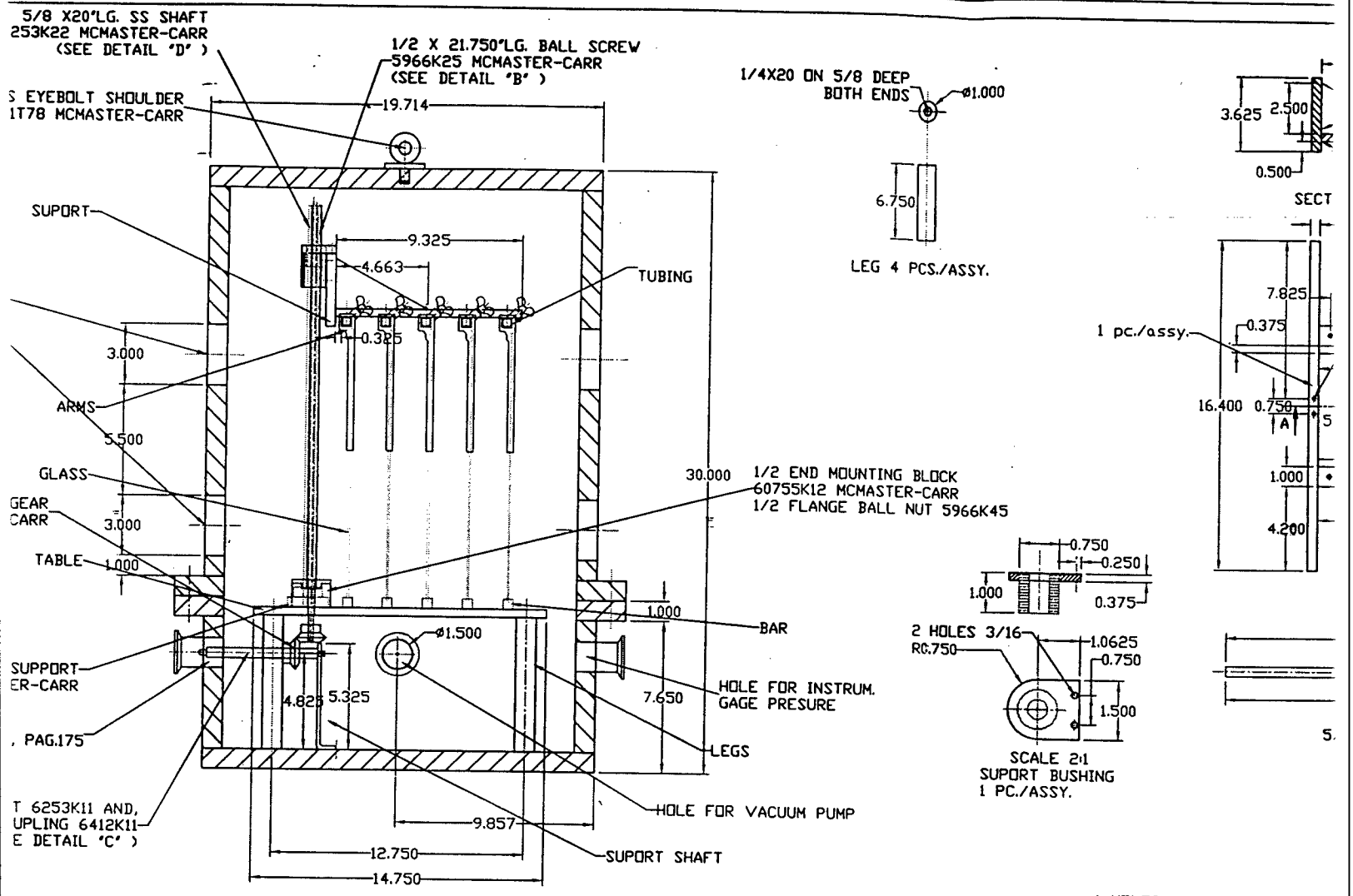
The following pages contain the mechanical drawings of the vacuum chamber designed to be used for filling the large MOE liquid crystal shutter cells.


5/8 X20"LG. SS SHAFT
6253K22 MCMASTER-CARR
(SEE DETAIL "D")
1/2 X 21.750"LG. BALL SCREW
5966K25 MCMASTER-CARR
(SEE DETAIL "B")
1/2-13 SS EYEBOLT SHOULDER
8891T78 MCMASTER-CARR
19.714
SUPORT
9.325
4.663
WINDOWS
TUBING
0.325
3.000
ARMS
5.500
GLASS
30.000
1/2" STEEL MITER GEAR
6529K16 MCMASTER-CARR
3.000
TABLE
1.000
1.000
Ø1.500
1/2 FLANGE MOUNT SUPPORT
57745K21 MCMASTER-CARR
4.825
5.325
7.650
HOLE FOR K200-BRM, PAG.175
6"X1/2 SS SHAFT 6253K11 AND,
1/4" STEEL COUPLING 6412K11
(SEE DETAIL "C")
HOLE FOR
9.857
SUPORT SHAFT
12.750
14.750

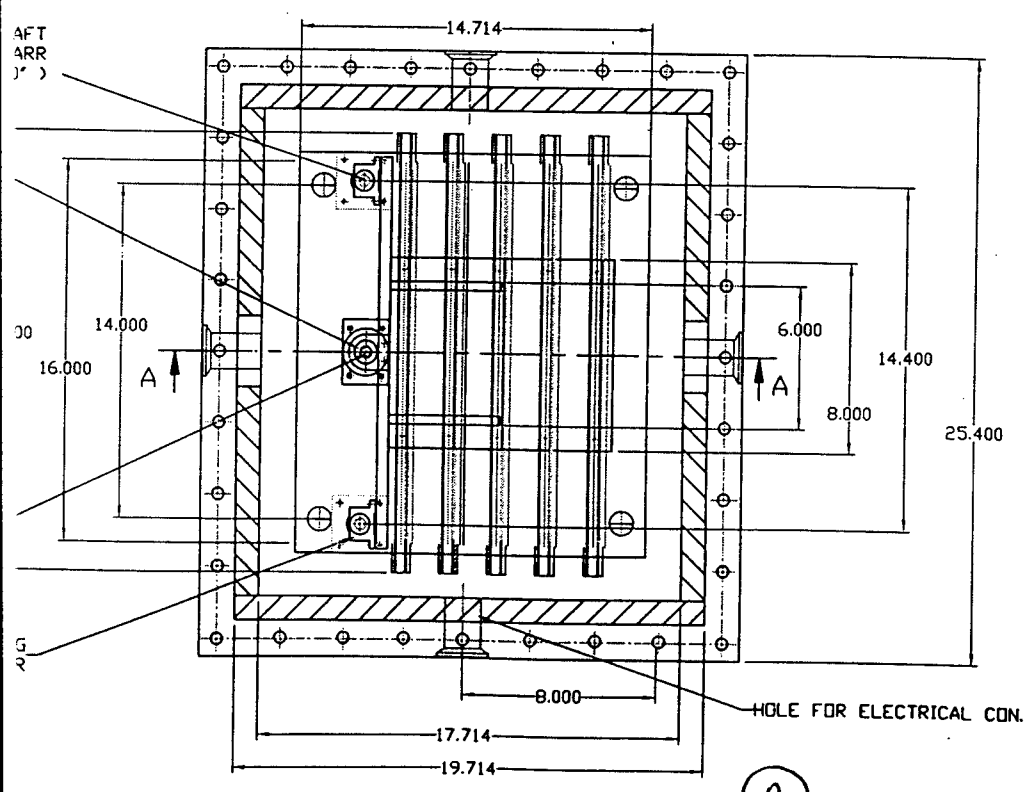
SECTION A-A


5/8 X20"LG. SS SHAFT
6253K22 MCMASTER-CARR
(SEE DETAIL "D")
14.714
1/2 BRONZE SLEEVE BEARING
6391K225 MCMASTER-CARR
18.400
14.000
16.000
A
A
6.000
8.000
1/2 X 21.750"LG. BALL SCREW
5966K25 MCMASTER-CARR
(SEE DETAIL "A")
5/8 PILLOW BLOCK BEARING
6255K341 MCMASTER-CARR
8.000
HOLE FOR EL.
17.714
19.714

(1)



SECTION A-A



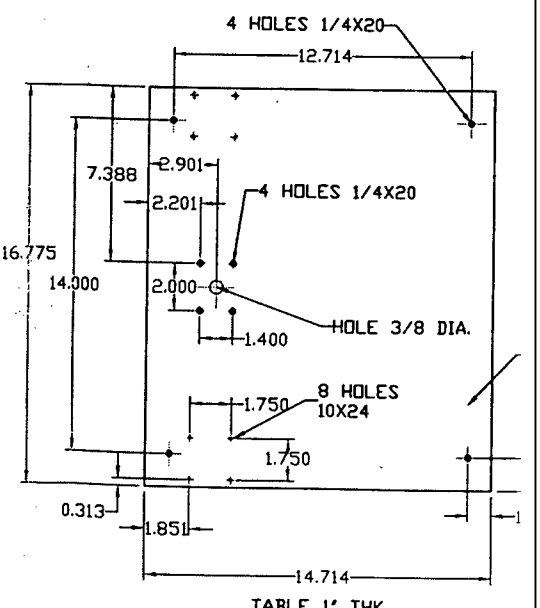
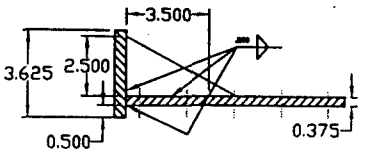


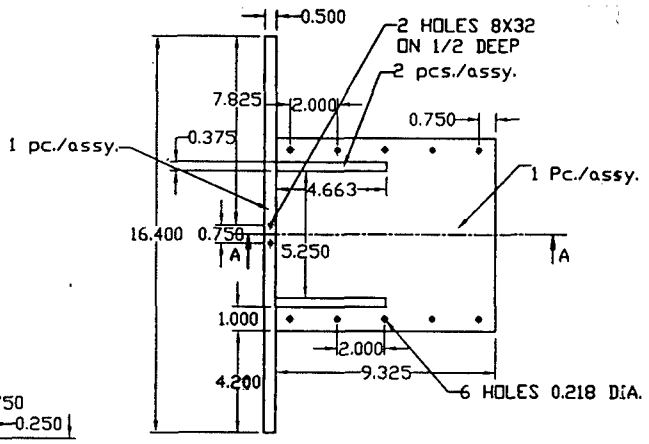
TABLE 1" THK.
1PC./ASSY.

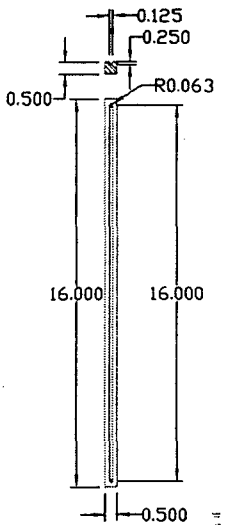
Rev	By	Date	Ap'd	Description
A	CP	24JUL98		INITIAL REL

100100-01 A

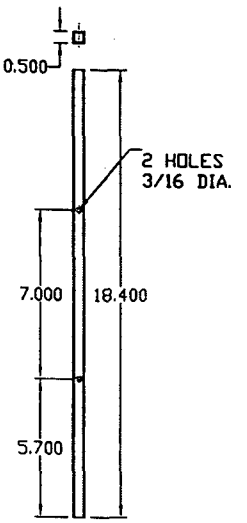


SECTION A-A

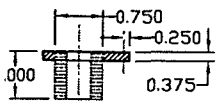


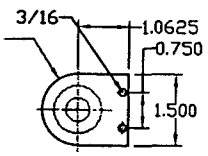


BAR 5 PCS./ASSY.

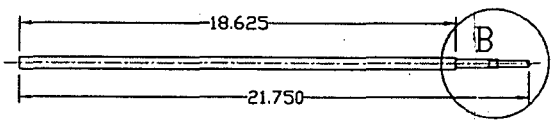


TUBING 5 PCS./ASSY.

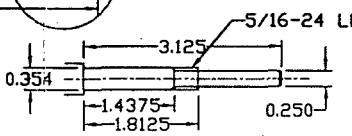




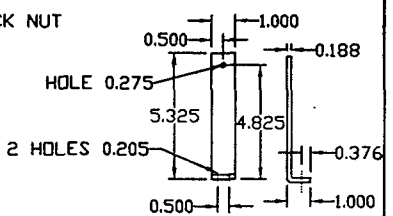
SCALE 2:1
SUPORT BUSHING
1 PC./ASSY.



5/8X21.750 BALL SCREW
1PC. /ASSY.



DETAIL (B)



SUPORT SHAFT 1PC/ASSY.

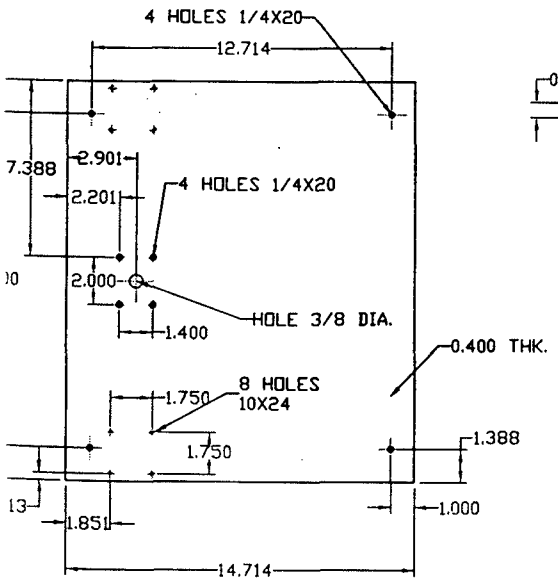
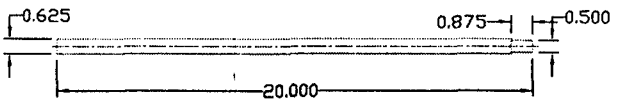
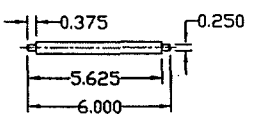


TABLE 1" THK.
1PC./ASSY.



SS GUIDE SHAFT
2PCS. /ASSY.
DETAIL 'D'



DETAIL 'C' SS SHAFT
1PC./ASSY.

Rev	By	Date	Ap'd	Description
A	CP	24JUL98		INITIAL RELEASE

3

Customer | Job Number

Det#	Qty	Material
		MATL1 ALUMINUM
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

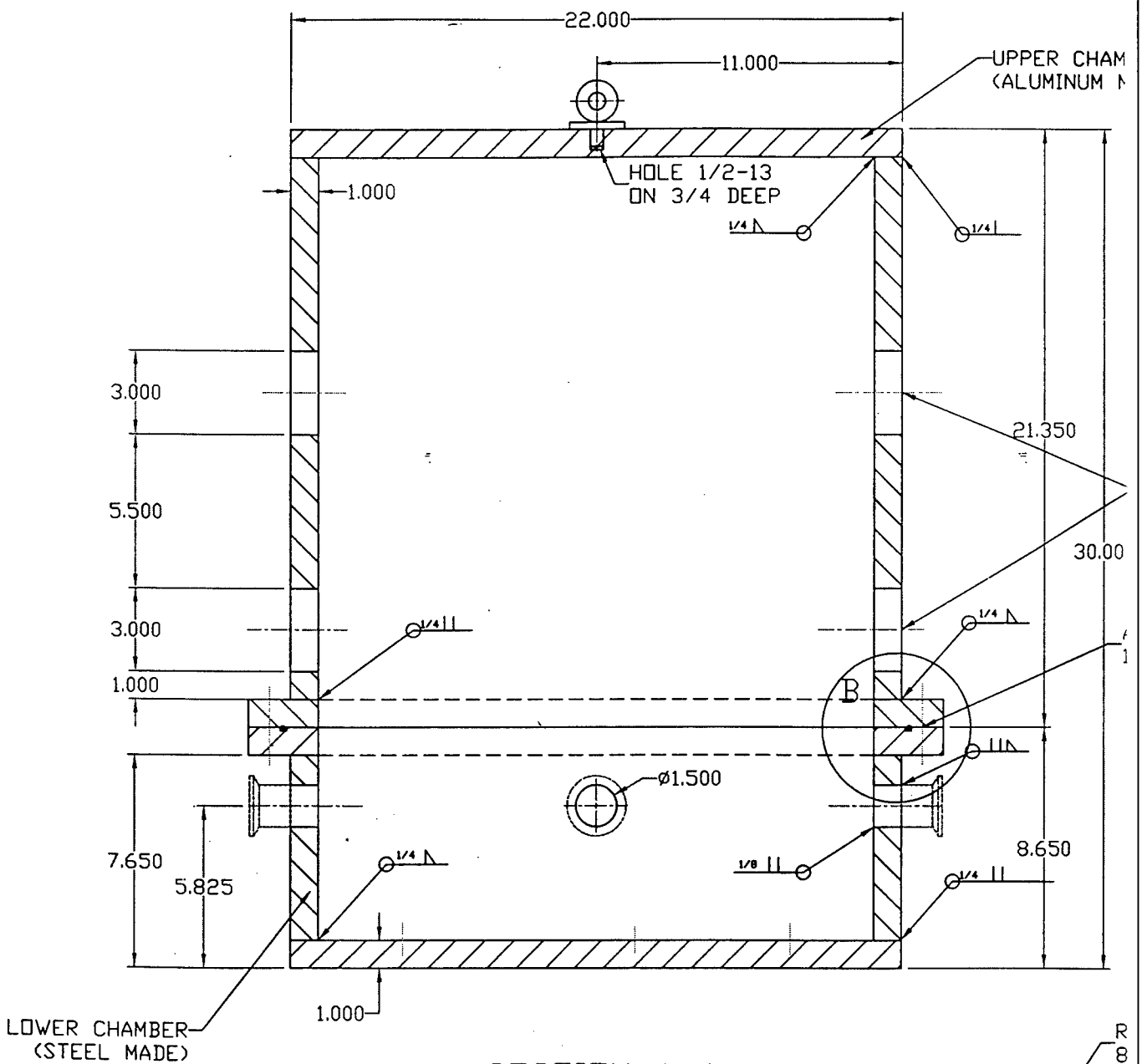
DIMENSIONAL MEDIA ASSOCAITES, INC.
New York, New York

125/ uos
Break sharp edges

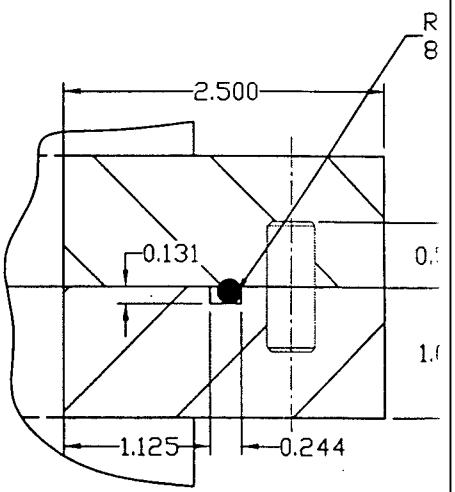
Drawn	CP	24JUL98
Check	CP	24JUL98
App'd	JB	24JUL98

VACUUM CHAMBER
DESIGN # 2

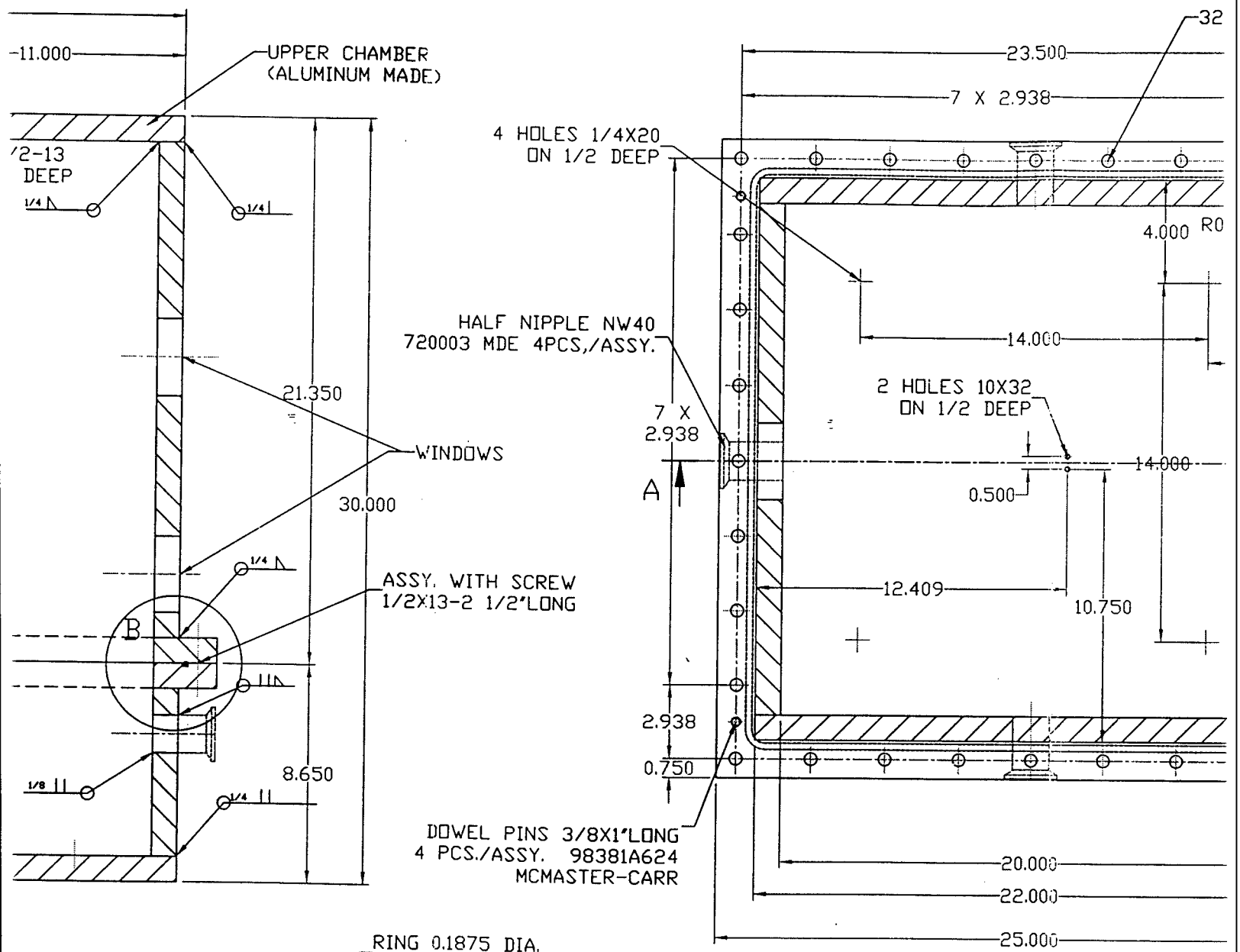
Scale	Drawing Number	Rev
1:1	100100-01	A

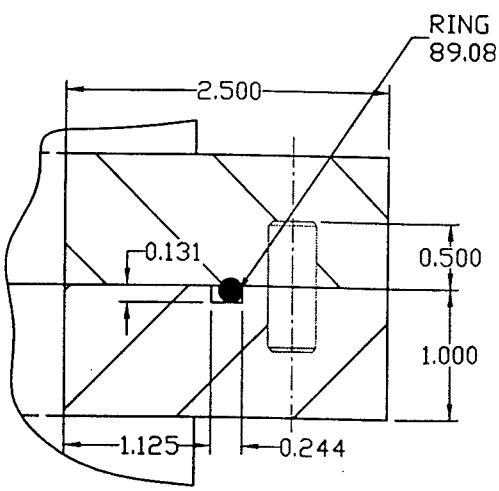


SECTION A-A



DETAIL (B)





DETAIL (B)

Rev	By	Date	Ap'd	Description
A	CP	4AUG98		INITIAL RELEASE

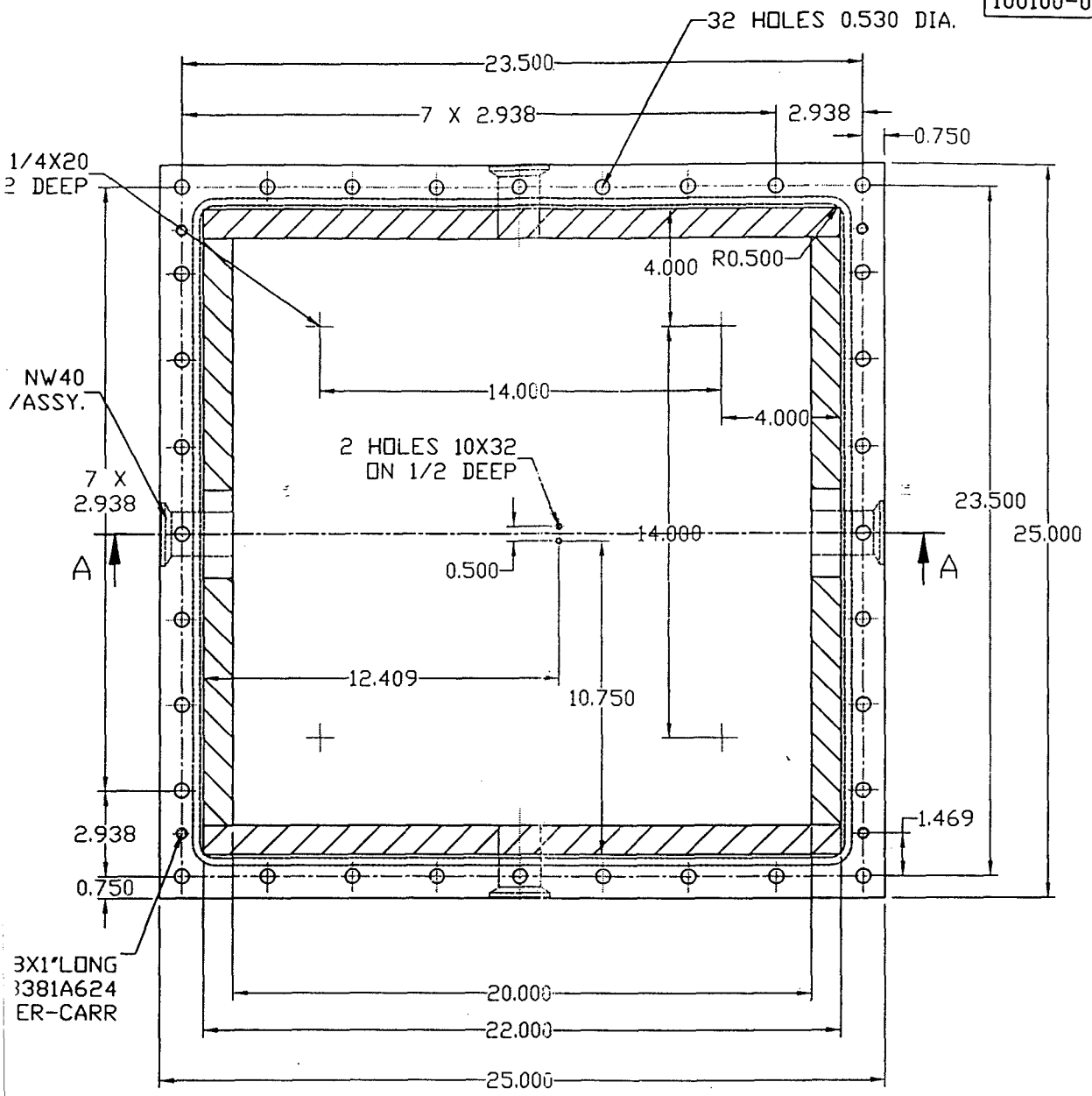
Customer

Job Number

Det#	Qty	Material
		MATL1
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

100100-01 | A



By	Date	Ap'd	Description
CP	4AUG98		INITIAL RELEASE

Customer | Job Number

Det #	Qty	Material
		MATL1
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

DIMENSIONAL MEDIA ASSOCAITES, INC.
New York, New York

125 µas
Break sharp edges

VACUUM CHAMBER
DESIGN #1

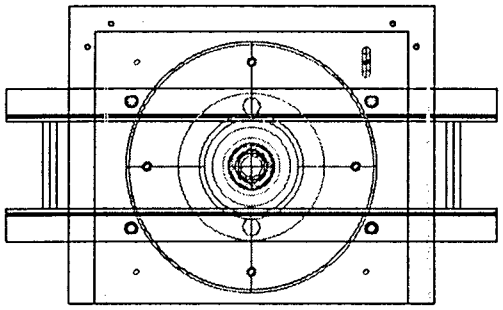
Drawn	CP	4AUG98
Check	CP	4AUG98
App'd	JB	4AUG98

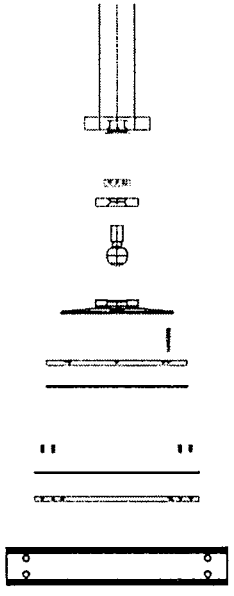
Scale	Drawing Number	Rev
1:1	100100-01	A

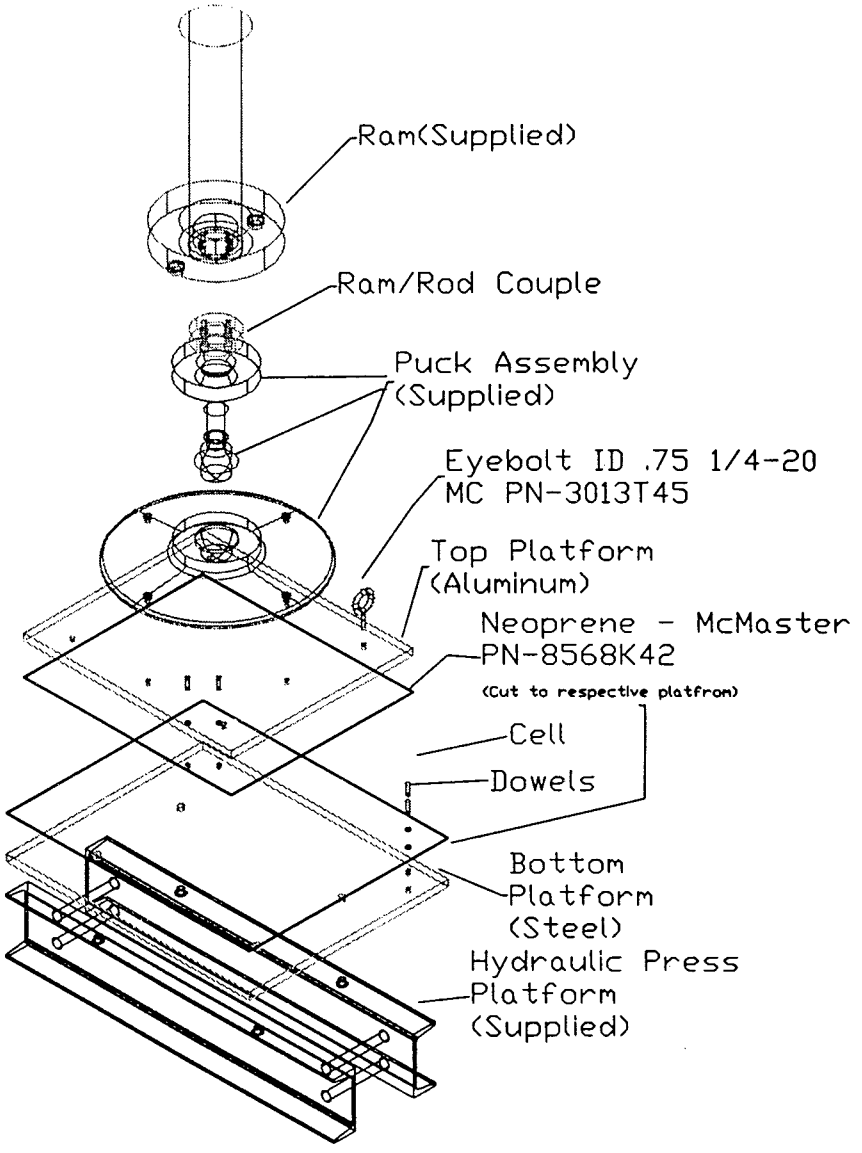
G. Cell press mechanical drawing

The following pages contain the mechanical drawings of the hydraulic cell press designed to press the MOE liquid crystal shutter cells down to a uniform thickness.

ASSEMBLY







Rev	By	Date	Description

Customer: CUST | Job Number: JOBNO

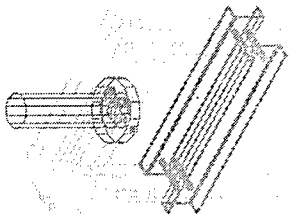
Detl #	Qty	Material
		MATL1
		MATL2
		MATL3

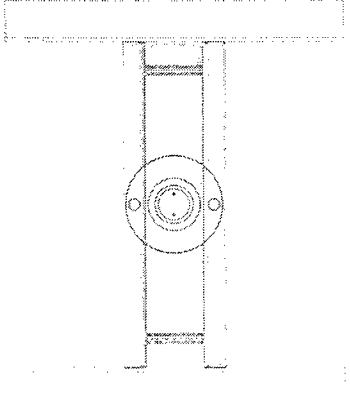
Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
XXX Decimals ±.010 .XX Decimals ±.020

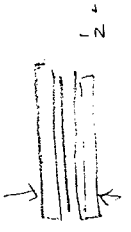
DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

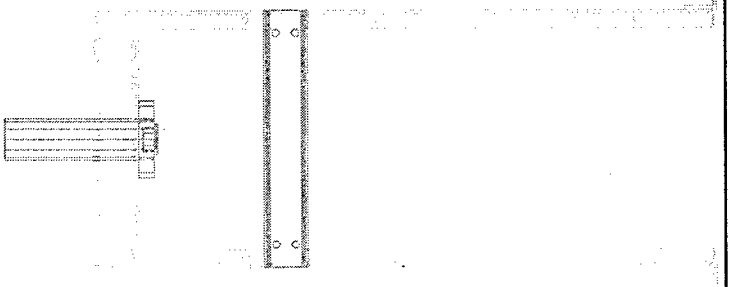
125 µin

Drawn	DK	DWNDATE	TITLE1 TITLE2 TITLE3
Check	CHK	CHKDATE	Scale / Drawing Number
App'd	REL	RELDATE	SCALE DWGNO

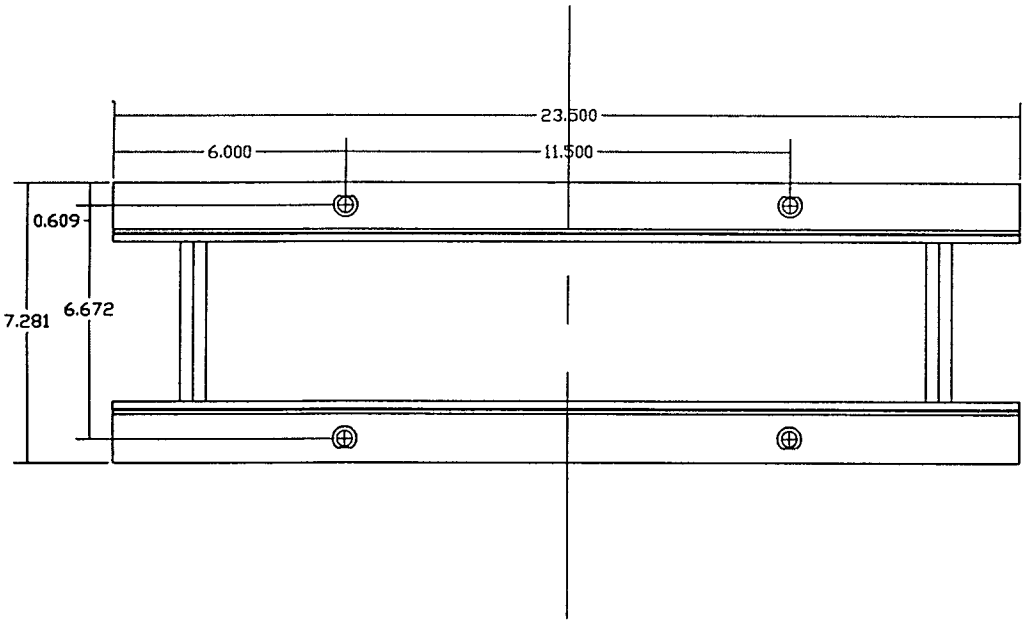


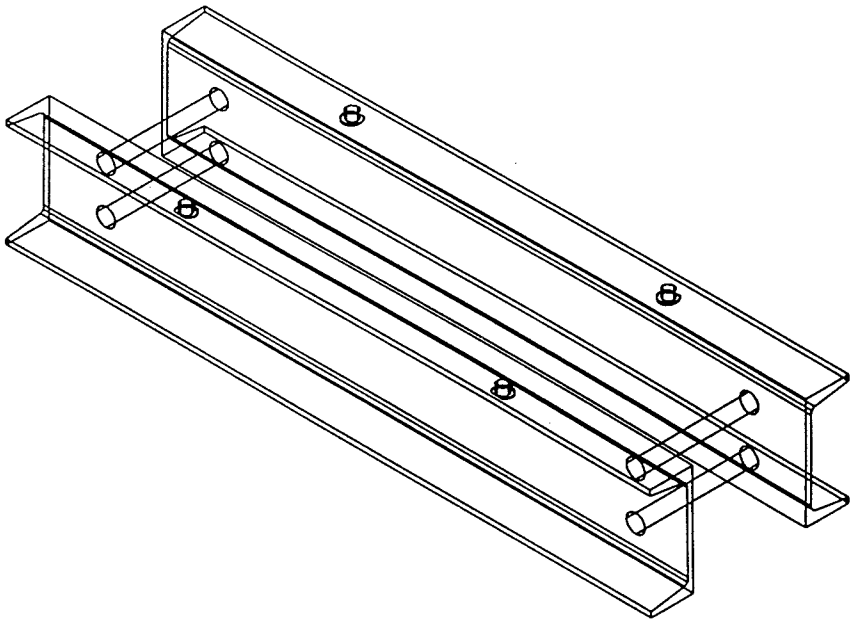


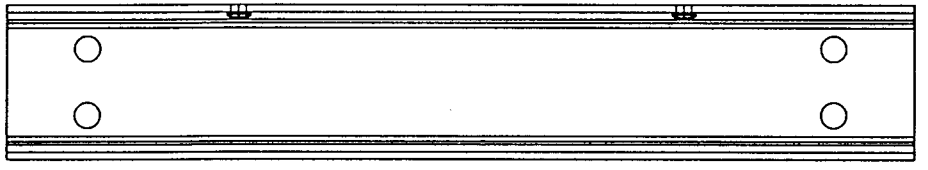


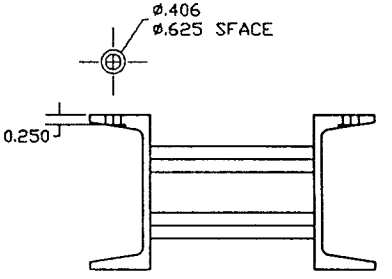


1003









Rev	By	Date	Description
			Drill and counter bore 4 holes. Match to corresponding 4 tapped holes on Dwg#1002. Part supplied.

Customer: CUST | Job Number: JOBNO

Detail	Qty	Material
		MATL1
		MATL2
		MATL3

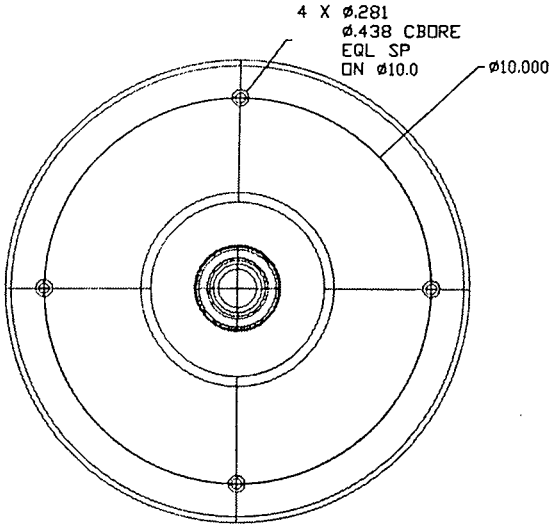
Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

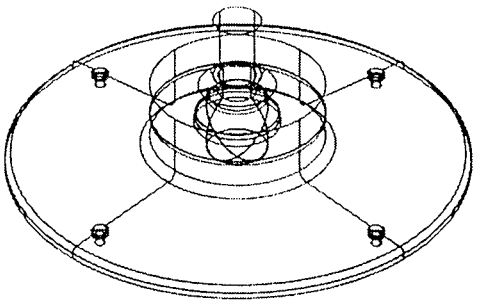
DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

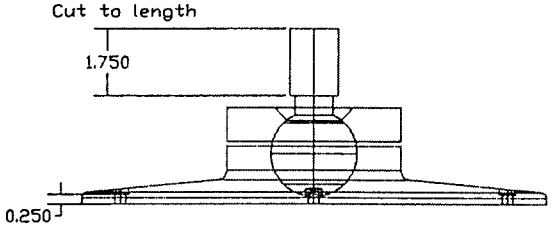
			PRESS_TABLE
			TITLE2
			TITLE3
Drawn	DK	DWNDATE	
Check	CHK	CHKDATE	
App'd	REL	RELDATE	

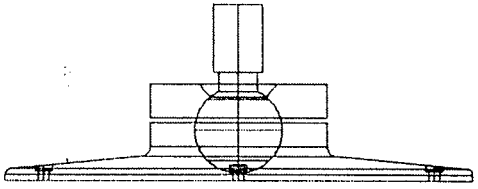
Scale	Drawing Number	Rev
SCALE	1003	

1005









Rev	By	Date	Description
			Drill & tap 4 holes. Cut rod to length.

Customer: CUST
Job Number: JOBNO

Dat#	Qty	Material
		MATL1
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles: ±30'
.XXX Decimals: ±.010 .XX Decimals: ±.020

DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

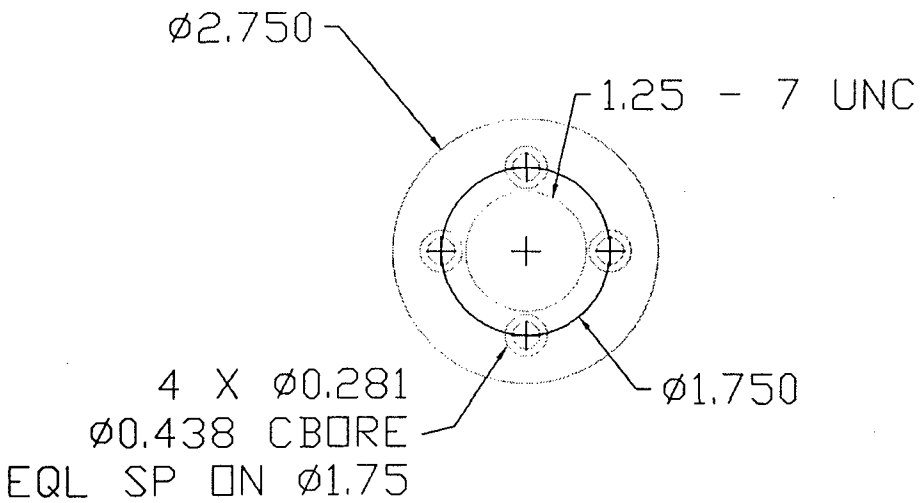
125 µin
Break sharp edges

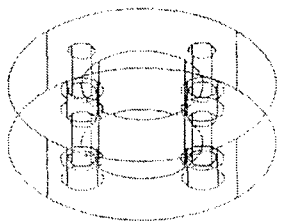
PUCK
TITLE2
TITLE3

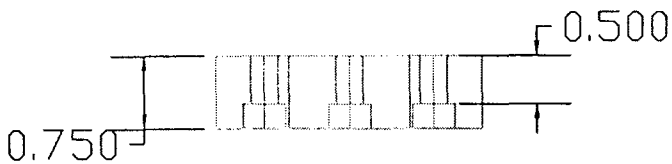
Drawn	DK	DRNDATE	
Check	CHK	CHKDATE	
App'd	REL	RELDATE	

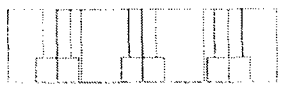
Scale	Drawing Number	Rev
SCALE	10005	

1006









Rev	By	Date	Description
			Steel

All information on this drawing is proprietary and confidential and must not be copied or disclosed without written authorization from Dimensional Media Associates, Inc. This document must be returned to Dimensional Media Associates, Inc. immediately upon request.

Customer: CUST
Job Number: JOBNO

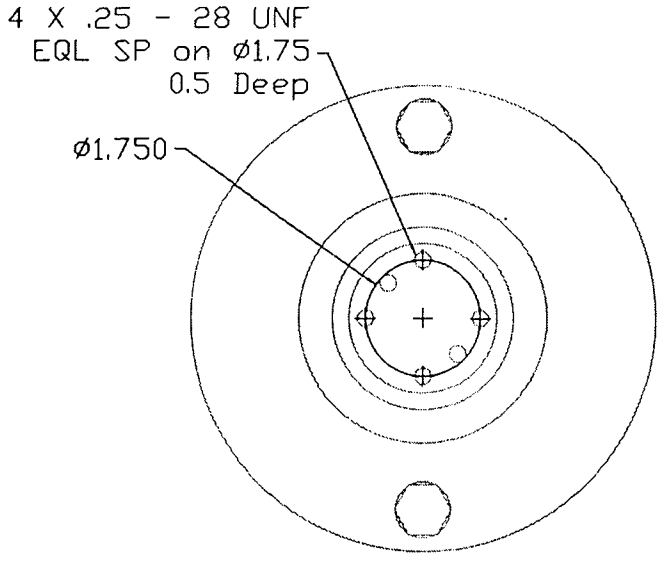
Dash	Qty	Material
		STEEL
		MATL2
		MATL3

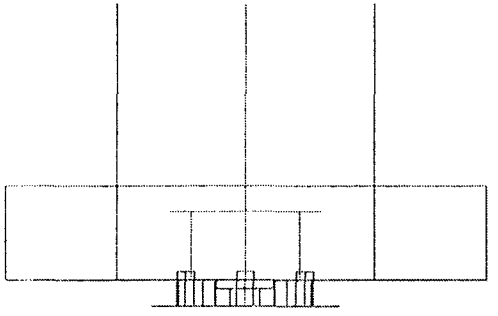
Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

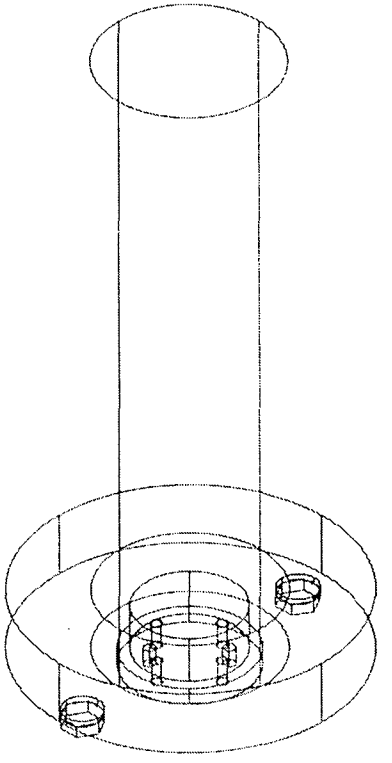
DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

Drawn	DK	DWNDATE	COUPLER
Check	CHK	CHKDATE	TITLE2
App'd	REL	RELDATE	TITLE3

Scale	Drawing Number	Rev
SCALE	1006	







Rev	By	Date	Description
			Modify to print. Cross section of cylinder assembly attatched. Supply screws.

Customer: CUST | Job Number: JOBNO

Def	Qty	Material
	4	0.25-28UNF_1"LONG_SCKTCAPSCREW
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

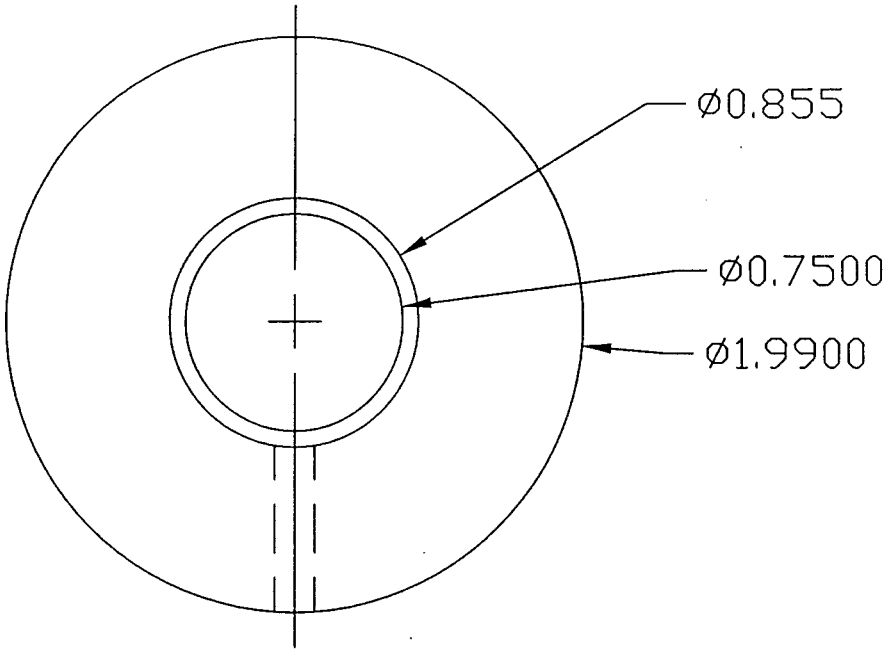
DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

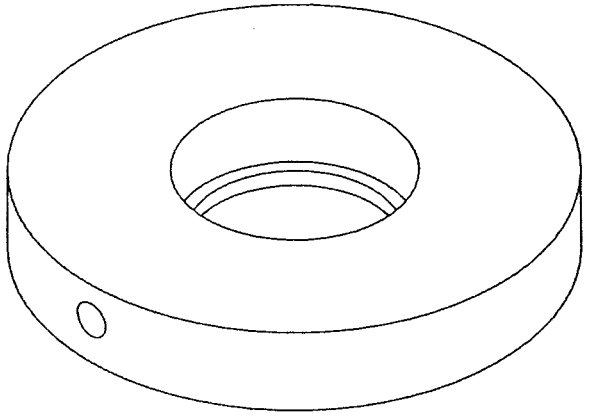
125/ USE — Break sharp edges

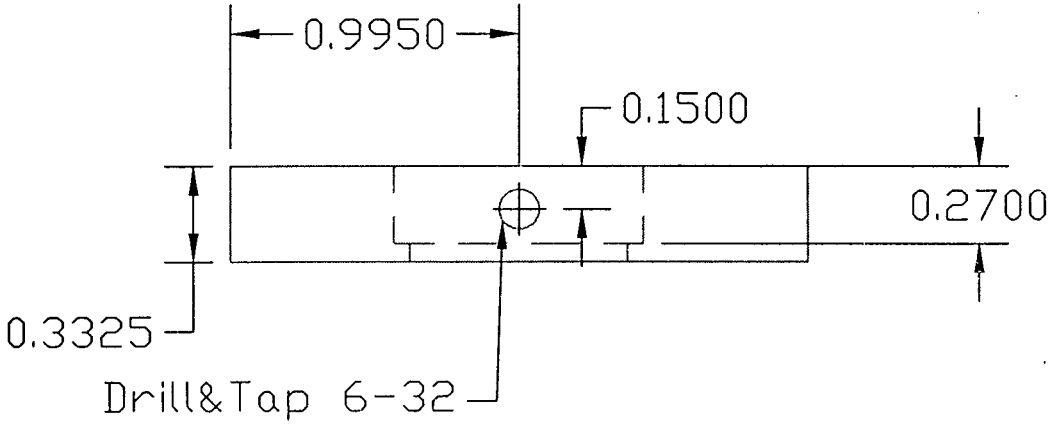
CYLINDER
TITLE2
TITLE3

Drawn	DK	DWNDATE	
Check	CHK	CHKDATE	
App'd	REL	RELDATE	

Scale	Drawing Number	Rev
SCALE	1007	







Rev	By	Date	Description

All information on this drawing is proprietary and confidential and must not be copied or disclosed without written authorization from Dimensional Media Associates, Inc. This document must be returned to Dimensional Media Associates, Inc. immediately upon request.

Customer: CUST | Job Number: JOBNO

Detail	Qty	Material
		ALUMINUM
		MATL2
		MATL3

Tolerances Unless Otherwise Noted:
Whole Nos. and Fractions: ±1/32 Angles ±30'
.XXX Decimals ±.010 .XX Decimals ±.020

DIMENSIONAL MEDIA ASSOCIATES, INC.
New York, New York

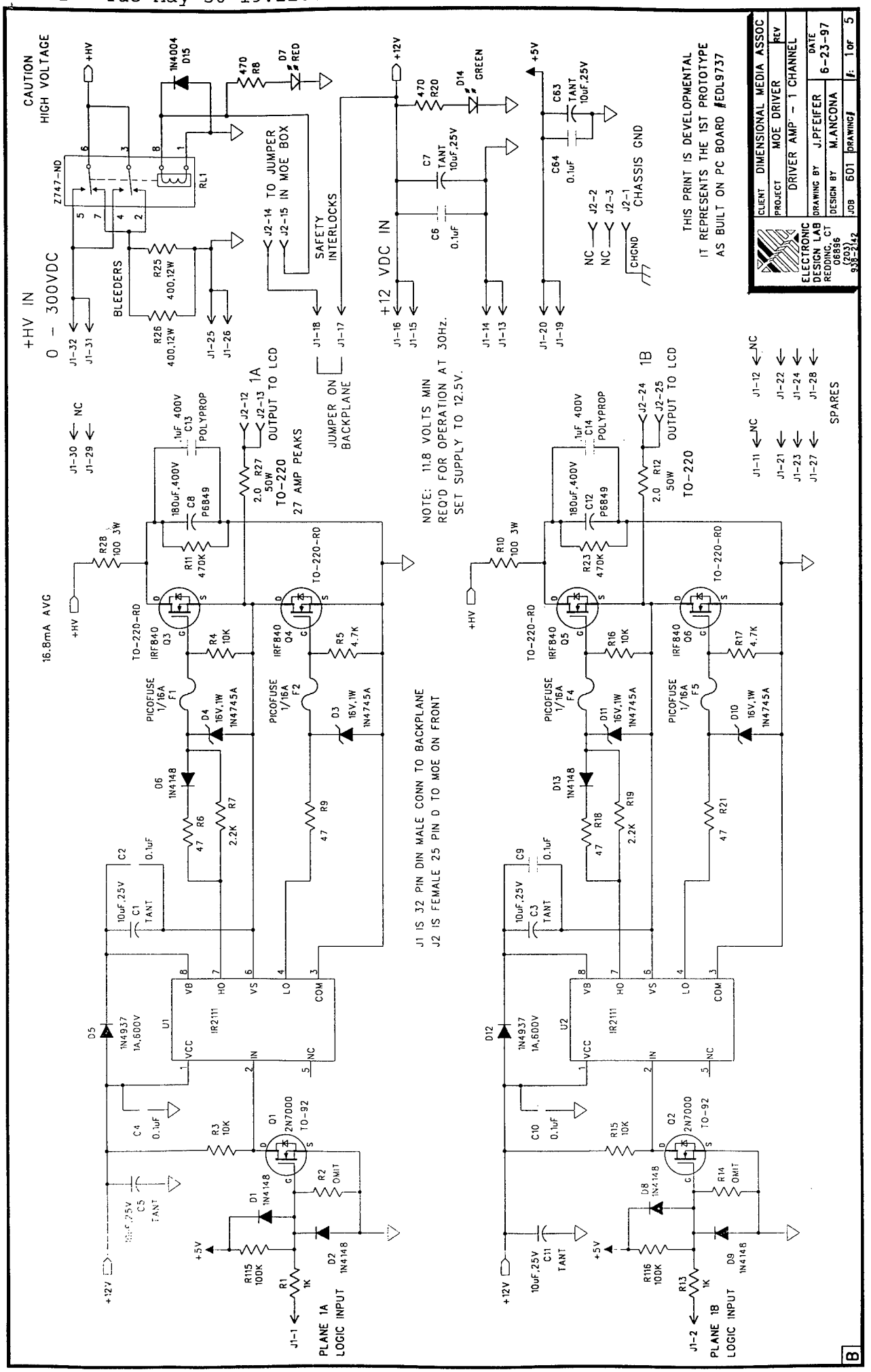
125/ uos — Break sharp edges

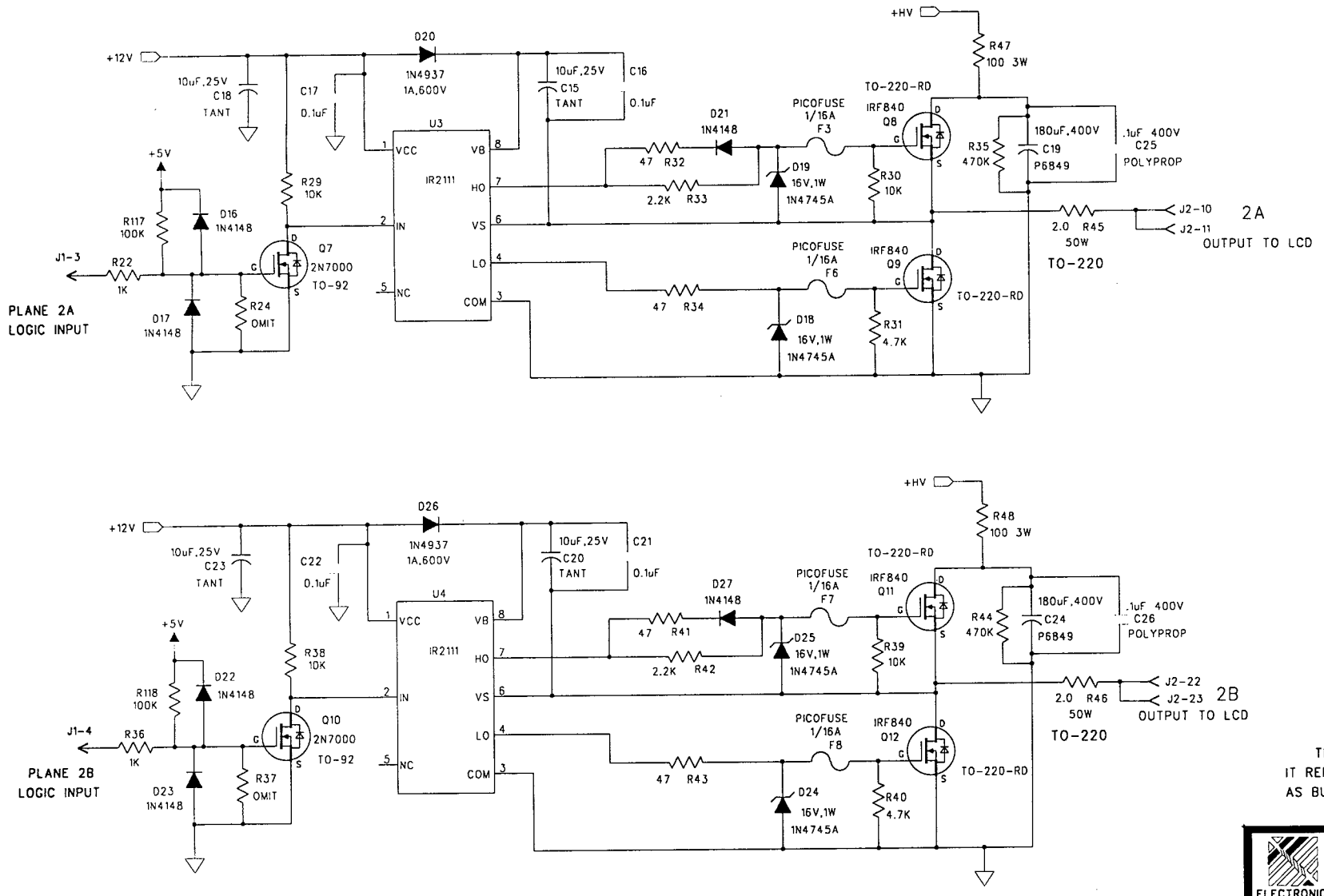
LENSMOUNT
DARPA02
TITLE3

Drawn	DK	2/1/98	Scale	Drawing Number	Rev
Check	CHK	CHKDATE			
App'd	REL	RELDATE	NOSCALE	2-001	

H. MOE Driver Schematics

The following pages contain the electronic schematics of the MVD framebuffer electronics.

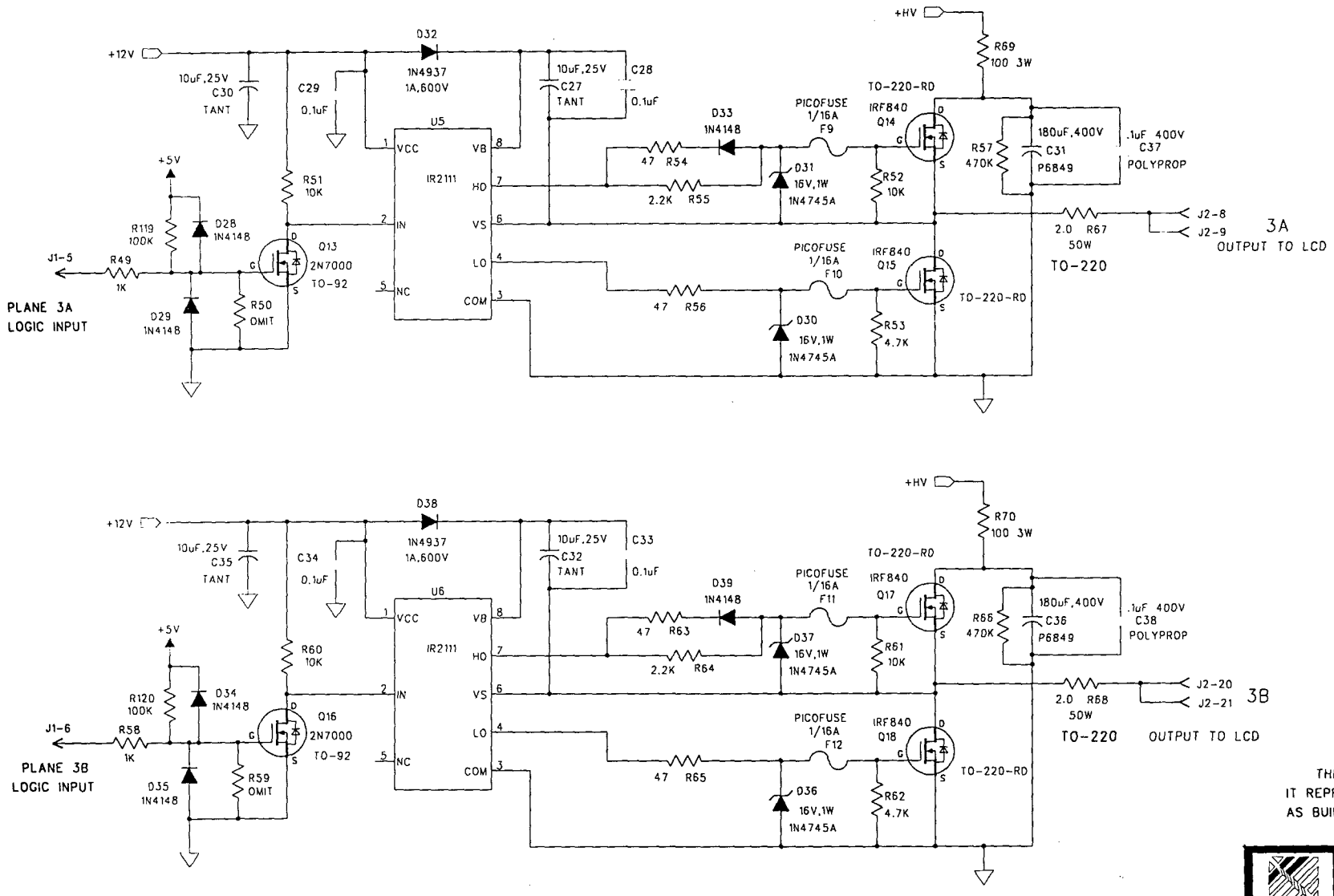

CAUTION
HIGH VOLTAGE
+HV IN
0 - 300VDC
+HV
1N4004
D15
470
R8
D7
RED
Z747-ND
RL1
BLEEDERS
R25
400,12W
R26
400,12W
J2-14 TO JUMPER
J2-15 IN MOE BOX
SAFETY
INTERLOCKS
+12 VDC IN
+12V
470
R20
D14
GREEN
C7
TANT
10uF,25V
C6
0.1uF
+5V
C63
TANT
10uF,25V
C64
0.1uF
J2-2
J2-3
J2-1
CHASSIS GND
NC
NC
CHGND
THIS PRINT IS DEVELOPMENTAL
IT REPRESENTS THE 1ST PROTOTYPE
AS BUILT ON PC BOARD #EDL9737
ELECTRONIC DESIGN LAB
REDDING, CT 06896
(203) 938-2142
CLIENT DIMENSIONAL MEDIA ASSOC
PROJECT MOE DRIVER REV
DRIVER AMP - 1 CHANNEL
DRAWING BY J.PFEIFER DATE 6-23-97
DESIGN BY M.ANCONA
JOB 601 DRAWING# F: 1 OF 5
J1-32
J1-31
J1-25
J1-26
J1-18
J1-17
J1-16
J1-15
J1-14
J1-13
J1-20
J1-19
J1-30 NC
J1-29
J2-12
J2-13 1A
OUTPUT TO LCD
2.0 R27
50W
TO-220
27 AMP PEAKS
JUMPER ON
BACKPLANE
NOTE: 11.8 VOLTS MIN
REQ'D FOR OPERATION AT 30Hz.
SET SUPPLY TO 12.5V.
J1-11 NC
J1-12 NC
J1-21
J1-22
J1-23
J1-24
J1-27
J1-28
SPARES
J2-24
J2-25 1B
OUTPUT TO LCD
2.0 R12
50W
TO-220
R10
100 3W
180uF,400V
C12
P6849
.1uF 400V
C14
POLYPROP
R23
470K
TO-220-RD
IRF840
Q5
Q6
R16
10K
R17
4.7K
PICOFUSE
1/16A
F4
F5
D11
D10
16V,1W
1N4745A
D13
1N4148
R18
4.7
R19
2.2K
R21
47
C9
0.1uF
10uF,25V
C3
TANT
D12
1N4937
1A,600V
U2
IR2111
VB
HO
VS
LO
COM
VCC
IN
NC
C10
0.1uF
R15
10K
Q2
2N7000
TO-92
D8
1N4148
R14
OMIT
C11
10uF,25V
TANT
R116
100K
R13
1K
D9
1N4148
J1-2
PLANE 1B
LOGIC INPUT
R28
100 3W
180uF,400V
C8
P6849
.1uF 400V
C13
POLYPROP
R11
470K
R27
2.0
50W
TO-220
27 AMP PEAKS
TO-220-RD
IRF840
Q3
Q4
R4
10K
R5
4.7K
PICOFUSE
1/16A
F1
F2
D4
D3
16V,1W
1N4745A
D6
1N4148
R6
47
R7
2.2K
R9
47
C2
0.1uF
10uF,25V
C1
TANT
D5
1N4937
1A,600V
U1
IR2111
VB
HO
VS
LO
COM
VCC
IN
NC
C4
0.1uF
R3
10K
Q1
2N7000
TO-92
D1
1N4148
R2
OMIT
C5
TANT
R115
100K
R1
1K
D2
1N4148
J1-1
PLANE 1A
LOGIC INPUT
16.8mA AVG
J1 IS 32 PIN DIN MALE CONN TO BACKPLANE
J2 IS FEMALE 25 PIN D TO MOE ON FRONT
B

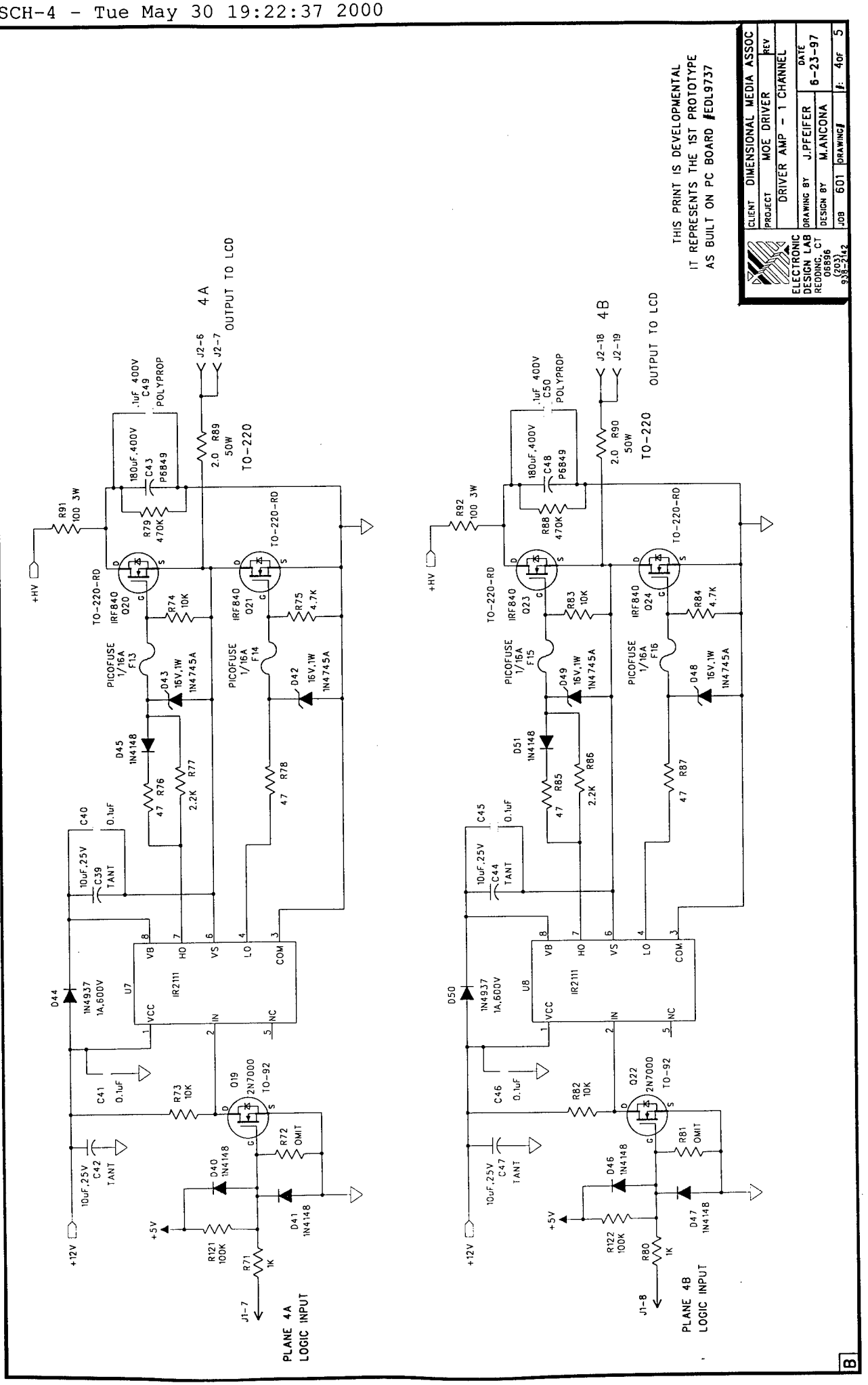


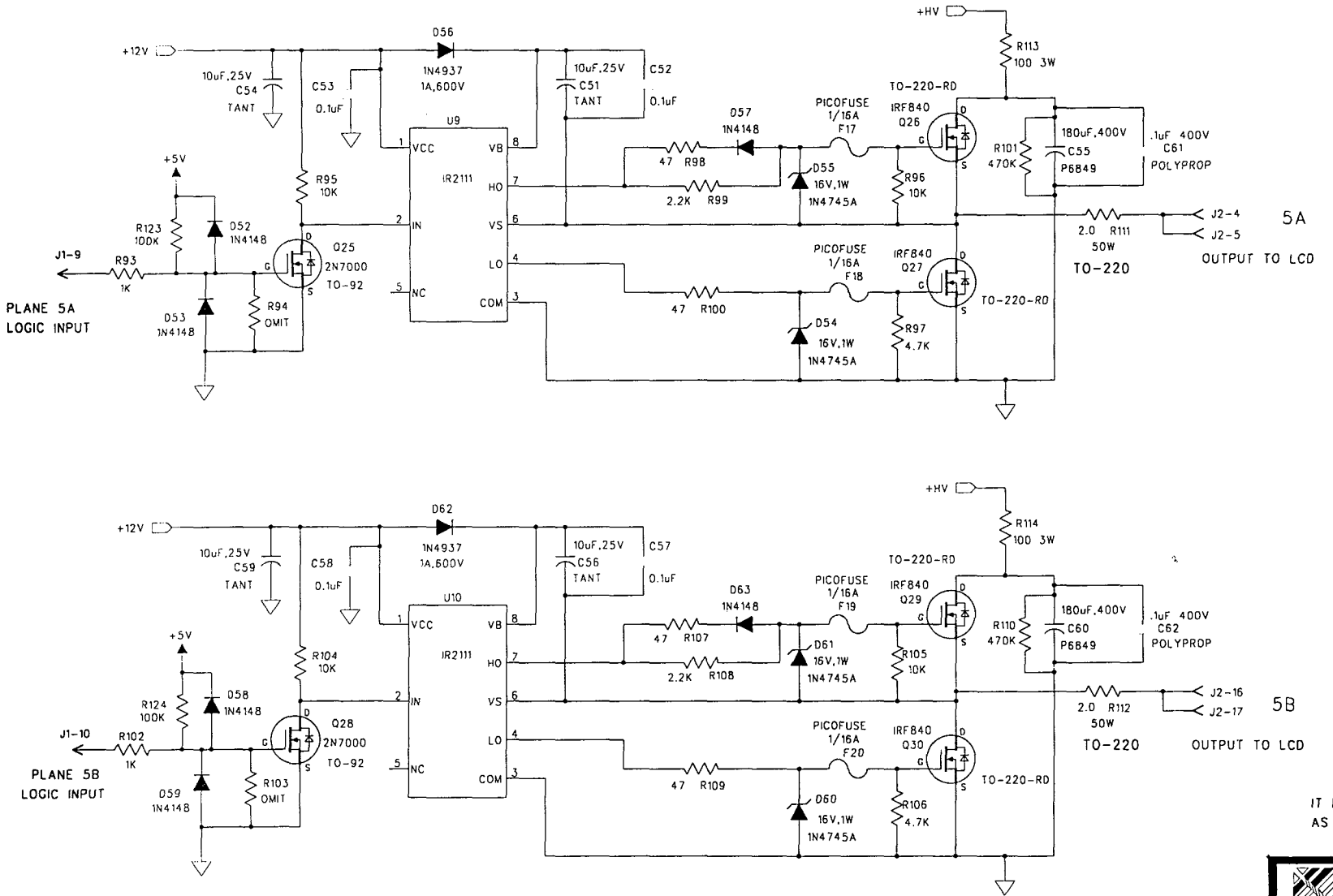
THIS PRINT IS DEVELOPMENTAL
IT REPRESENTS THE 1ST PROTOTYPE
AS BUILT ON PC BOARD #EDL9737

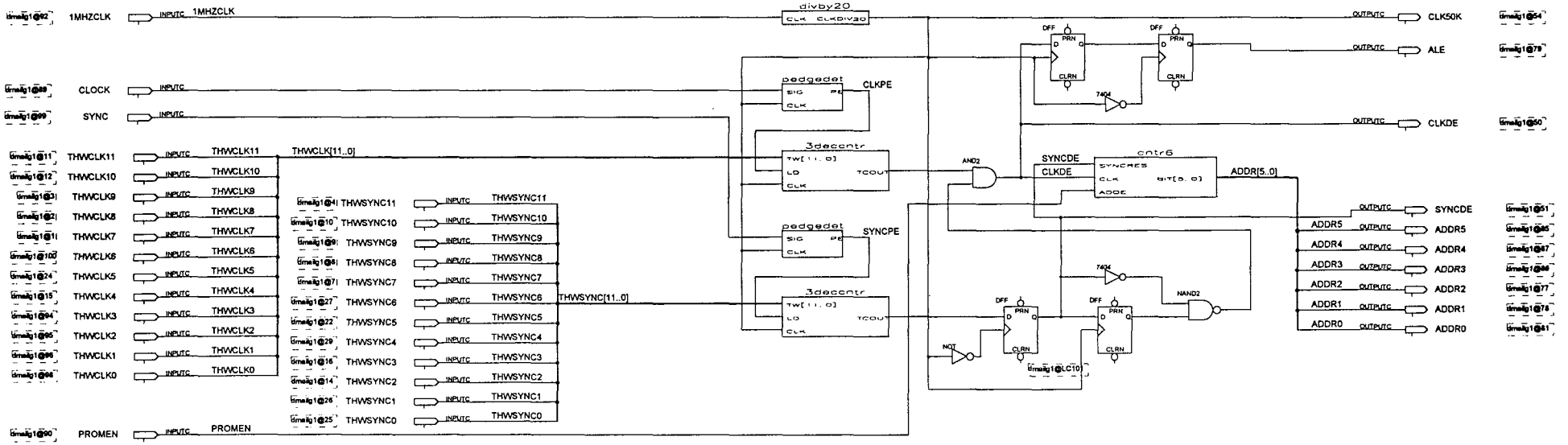
ELECTRONIC DESIGN LAB REDDING, CT 06896 (203) 938-2142	CLIENT	DIMENSIONAL MEDIA ASSOC	
	PROJECT	MOE DRIVER	REV
		DRIVER AMP - 1 CHANNEL	
	DRAWING BY	J.PFEIFER	DATE 6-23-97
	DESIGN BY	M.ANCONA	
	JOB 601	DRAWING#	#: 2OF 5

B

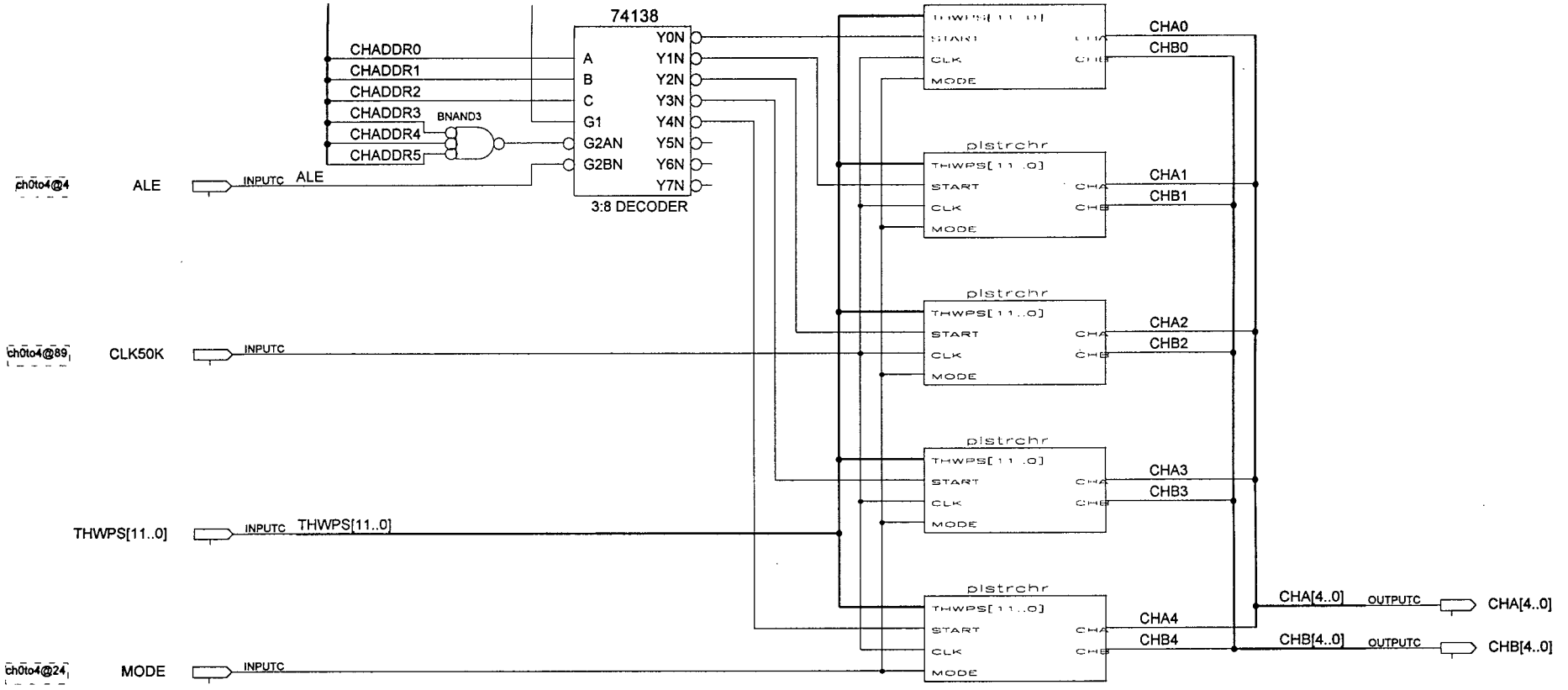

+12V
10uF,25V
C30
TANT
C29
0.1uF
D32
1N4937
1A,600V
U5
IR2111
VCC
VB
HO
VS
IN
LO
NC
COM
10uF,25V
C27
TANT
C28
0.1uF
+HV
R69
100 3W
TO-220-RD
D33
1N4148
PICOFUSE
1/16A
F9
IRF840
Q14
47 R54
2.2K R55
D31
16V,1W
1N4745A
R52
10K
R57
470K
180uF,400V
C31
P6849
.1uF 400V
C37
POLYPROP
+5V
R51
10K
R119
100K
D28
1N4148
Q13
2N7000
TO-92
J1-5
R49
1K
PLANE 3A
LOGIC INPUT
D29
1N4148
R50
OMIT
PICOFUSE
1/16A
F10
IRF840
Q15
47 R56
D30
16V,1W
1N4745A
R53
4.7K
2.0 R67
50W
TO-220
J2-8
J2-9
3A
OUTPUT TO LCD
+12V
10uF,25V
C35
TANT
C34
0.1uF
D38
1N4937
1A,600V
U6
IR2111
10uF,25V
C32
TANT
C33
0.1uF
+HV
R70
100 3W
TO-220-RD
D39
1N4148
PICOFUSE
1/16A
F11
IRF840
Q17
47 R63
2.2K R64
D37
16V,1W
1N4745A
R61
10K
R66
470K
180uF,400V
C36
P6849
.1uF 400V
C38
POLYPROP
+5V
R60
10K
R120
100K
D34
1N4148
Q16
2N7000
TO-92
J1-6
R58
1K
PLANE 3B
LOGIC INPUT
D35
1N4148
R59
OMIT
PICOFUSE
1/16A
F12
IRF840
Q18
47 R65
D36
16V,1W
1N4745A
R62
4.7K
2.0 R68
50W
TO-220
J2-20
J2-21
3B
OUTPUT TO LCD
THIS PRINT IS DEVELOPMENTAL
IT REPRESENTS THE 1ST PROTOTYPE
AS BUILT ON PC BOARD #EDL9737
ELECTRONIC
DESIGN LAB
REDDING, CT
06896
(203)
938-2142
CLIENT DIMENSIONAL MEDIA ASSOC
PROJECT MOE DRIVER
REV
DRIVER AMP - 1 CHANNEL
DRAWING BY J.PFEIFER
DATE
6-23-97
DESIGN BY M.ANCONA
JOB 601
DRAWING#
#: 3OF 5
B


THIS PRINT IS DEVELOPMENTAL
IT REPRESENTS THE 1ST PROTOTYPE
AS BUILT ON PC BOARD #EDL9737
CLIENT DIMENSIONAL MEDIA ASSOC
PROJECT MOE DRIVER
REV
DRIVER AMP - 1 CHANNEL
ELECTRONIC DESIGN LAB
REDDING, CT 06896
DRAWING BY J.PFEIFER
DESIGN BY M.ANCONA
DATE 6-23-97
JOB 601
DRAWING#
SHEET 4 OF 5
4A
OUTPUT TO LCD
J2-6
J2-7
TO-220
2.0 R89 50W
.1uF 400V C49 POLYPROP
180uF,400V C43 P6849
R79 470K
R91 100 3W
+HV
TO-220-RD
IRF840 Q20
R74 10K
R75 4.7K
IRF840 Q21
PICOFUSE 1/16A F13
PICOFUSE 1/16A F14
D43 16V,1W 1N4745A
D42 16V,1W 1N4745A
D45 1N4148
R76 47
R77 2.2K
R78 47
C40 0.1uF
10uF,25V C39 TANT
D44 1N4937 1A,600V
U7 IR2111
VB HO VS LO COM VCC IN NC
C41 0.1uF
R73 10K
Q19 2N7000 TO-92
R72 OMIT
D40 1N4148
D41 1N4148
10uF,25V C42 TANT
+12V
+5V
R121 100K
R71 1K
J1-7
PLANE 4A
LOGIC INPUT
4B
OUTPUT TO LCD
J2-18
J2-19
TO-220
2.0 R90 50W
.1uF 400V C50 POLYPROP
180uF,400V C48 P6849
R88 470K
R92 100 3W
+HV
TO-220-RD
IRF840 Q23
R83 10K
R84 4.7K
IRF840 Q24
PICOFUSE 1/16A F15
PICOFUSE 1/16A F16
D49 16V,1W 1N4745A
D48 16V,1W 1N4745A
D51 1N4148
R85 47
R86 2.2K
R87 47
C45 0.1uF
10uF,25V C44 TANT
D50 1N4937 1A,600V
U8 IR2111
VB HO VS LO COM VCC IN NC
C46 0.1uF
R82 10K
Q22 2N7000 TO-92
R81 OMIT
D46 1N4148
D47 1N4148
10uF,25V C47 TANT
+12V
+5V
R122 100K
R80 1K
J1-8
PLANE 4B
LOGIC INPUT
B

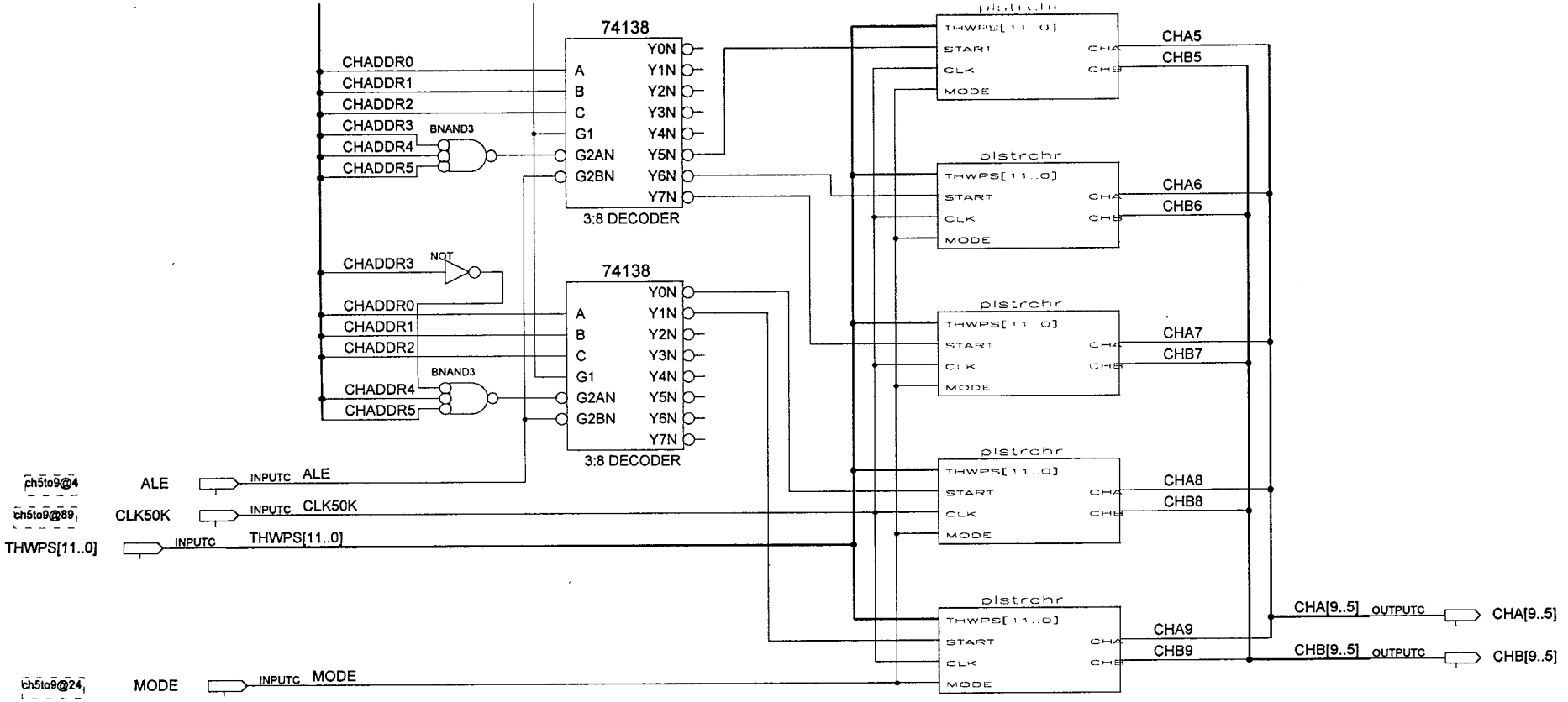




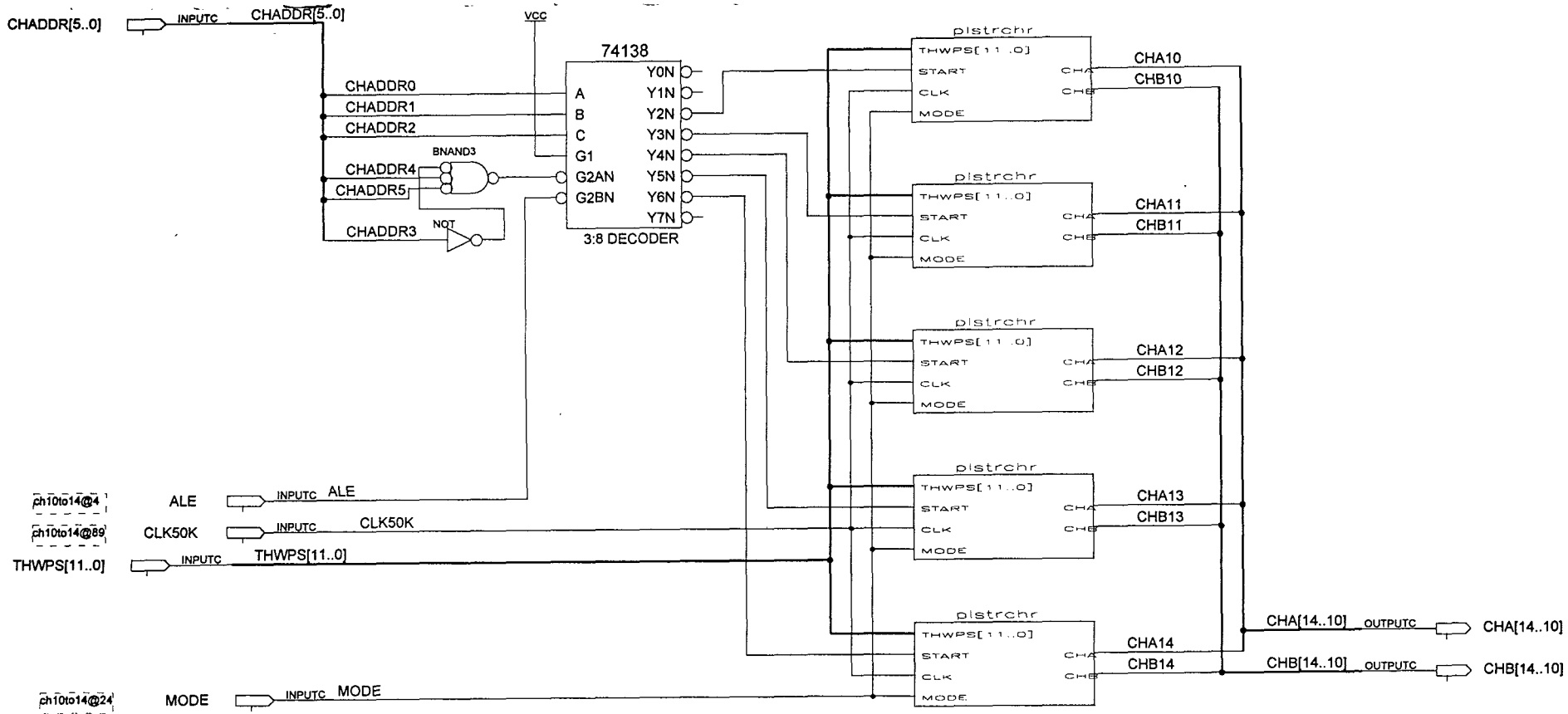
TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronics Design Lab		
SIZE D	NUMBER INPUT LOGIC	REV A	
DATE 10:06a 10-31-1997	SHEET 1&2 OF 6		



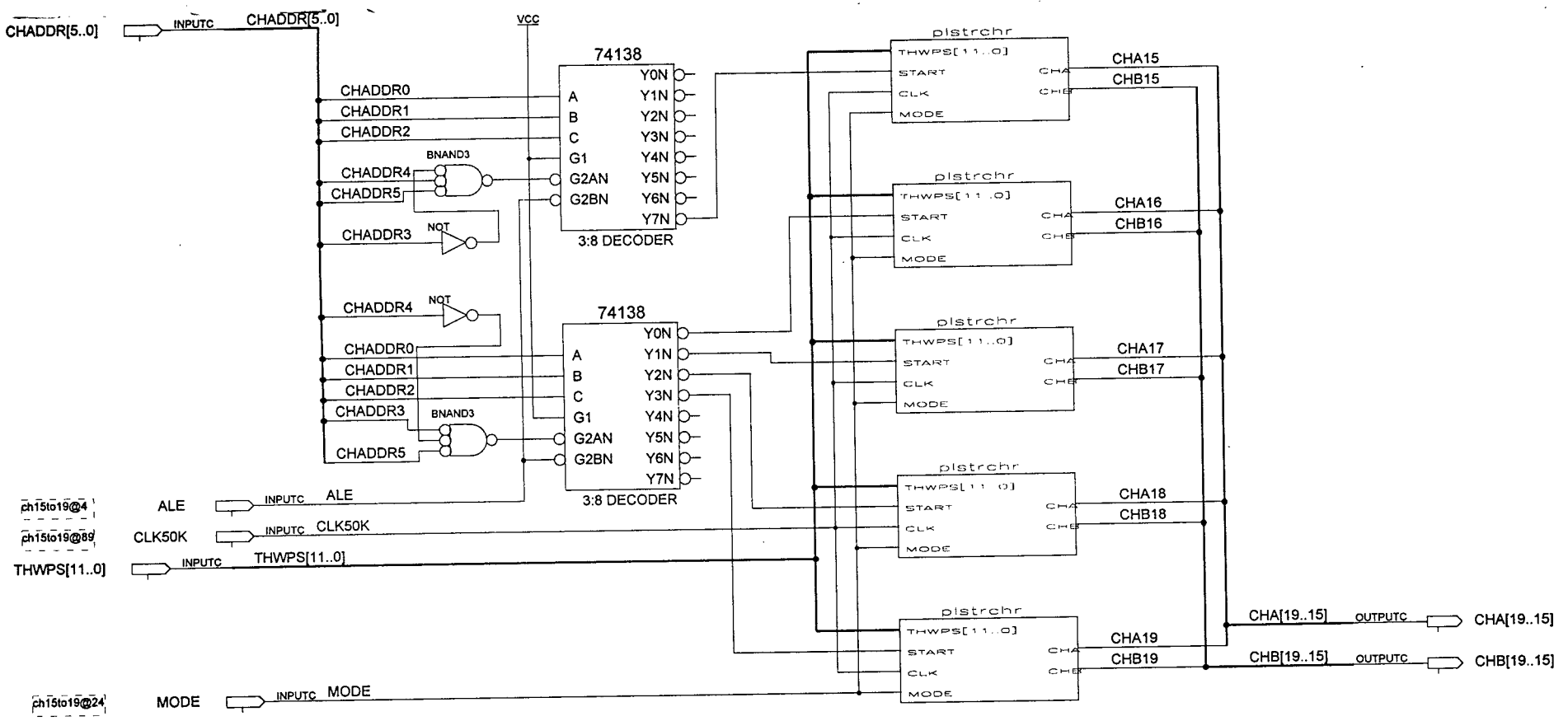
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 0 to 4	REV A	
DATE 9:39a 6-26-1997	SHEET 1	OF 3	



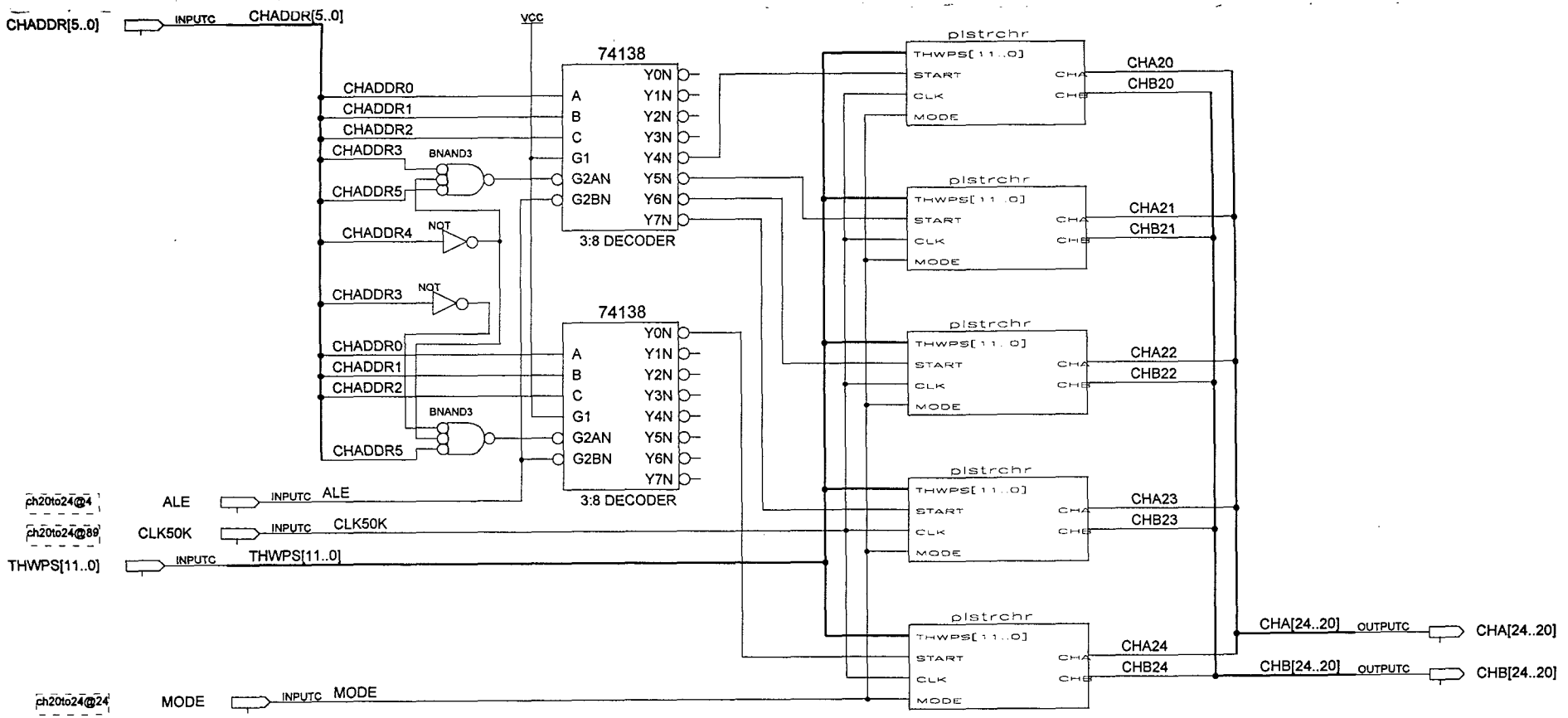
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 5 to 9	REV A	
DATE 10:00a 6-26-1997	SHEET 1	OF 3	



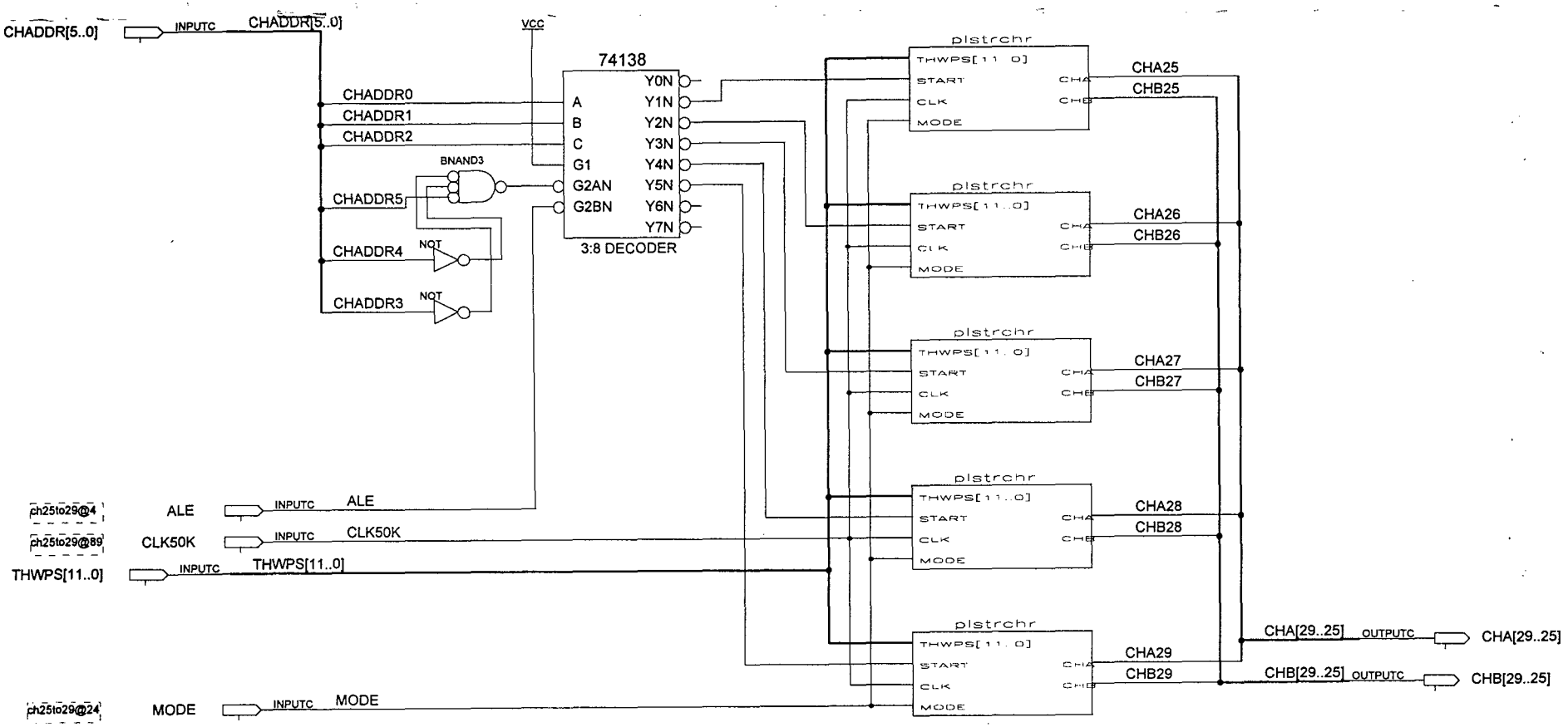
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 10 to 14	REV A	
DATE 10:22a 6-26-1997		SHEET 1	OF 3



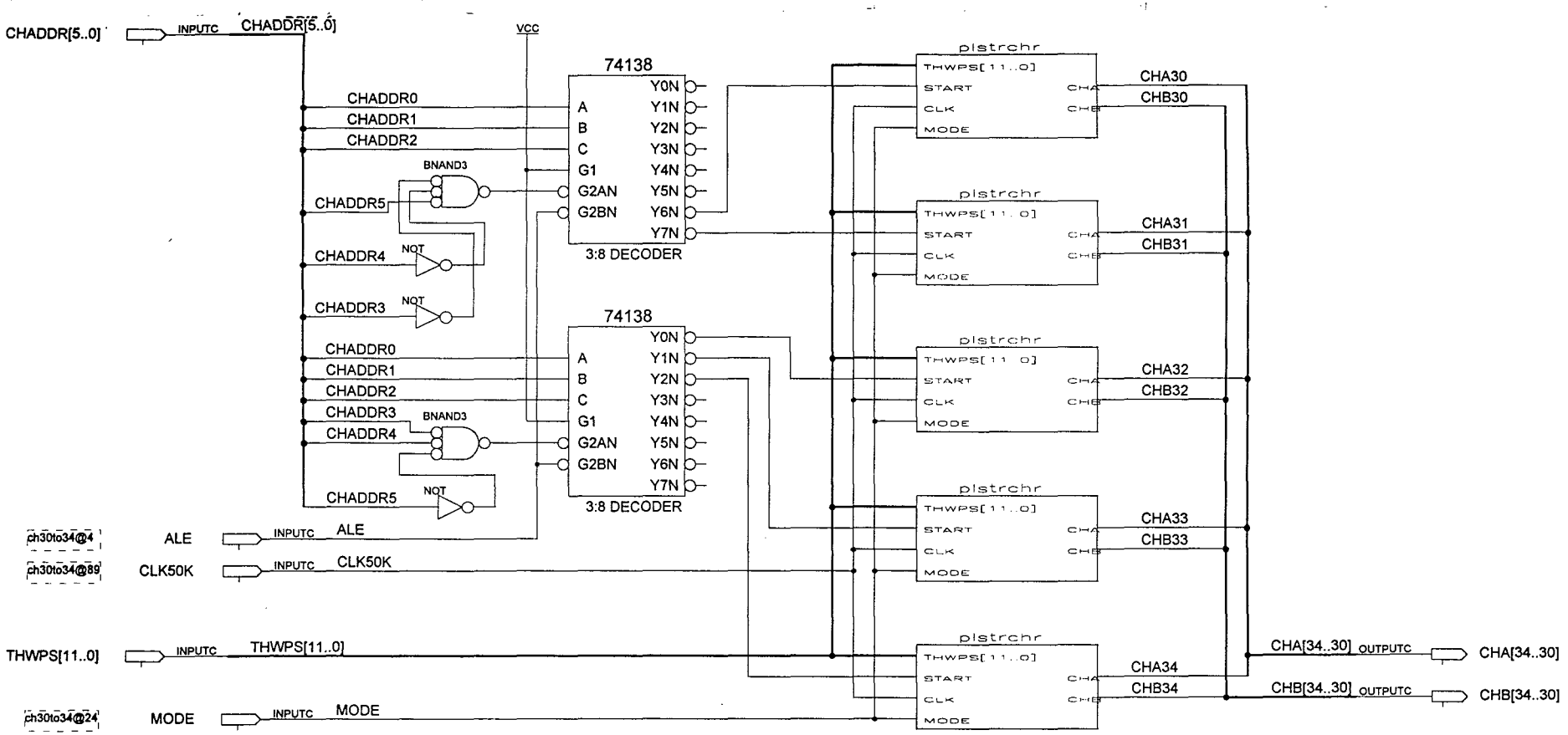
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 15 to 19	REV	A
DATE 10:27a 6-26-1997	SHEET 1	OF	3



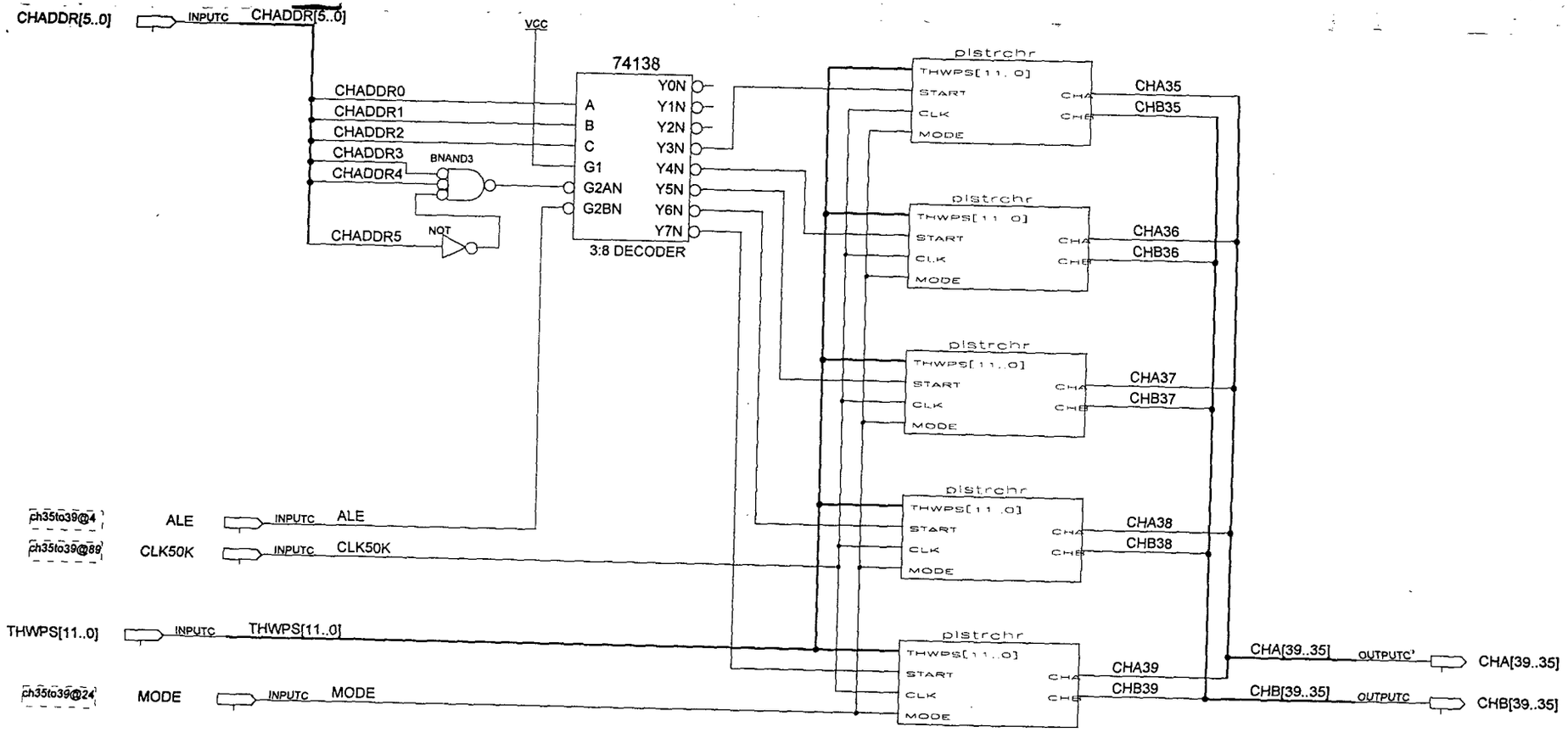
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 20 to 24	REV	A
DATE 10:29a 6-26-1997	SHEET 1	OF	3



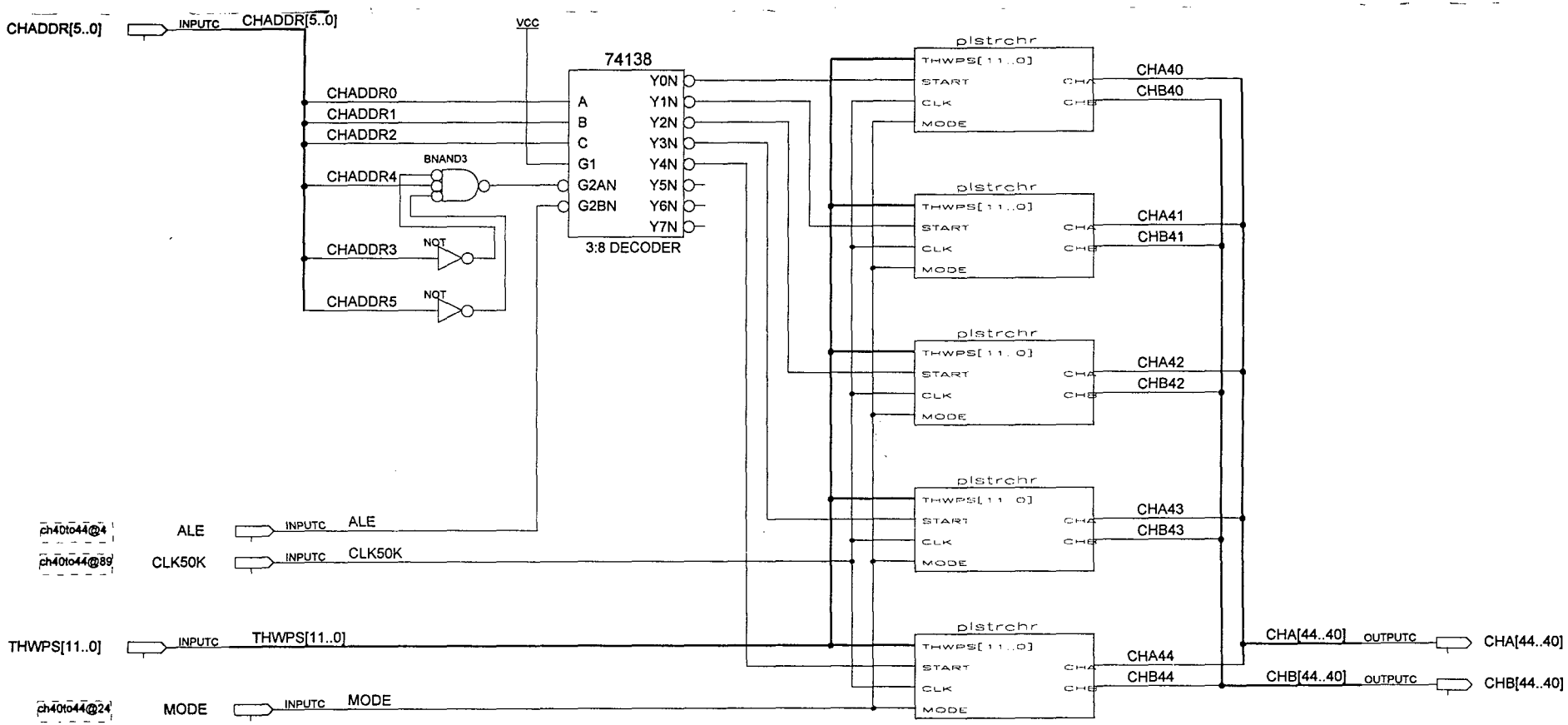
TITLE	LCD DRIVER ELECTRONICS FOR MOE			
COMPANY	Dimensional Media Associates			
DESIGNER	Electronic Design Lab			
SIZE C		NUMBER Channels 25 to 29	REV	A
DATE 10:37a 6-26-1997			SHEET 1	OF 3



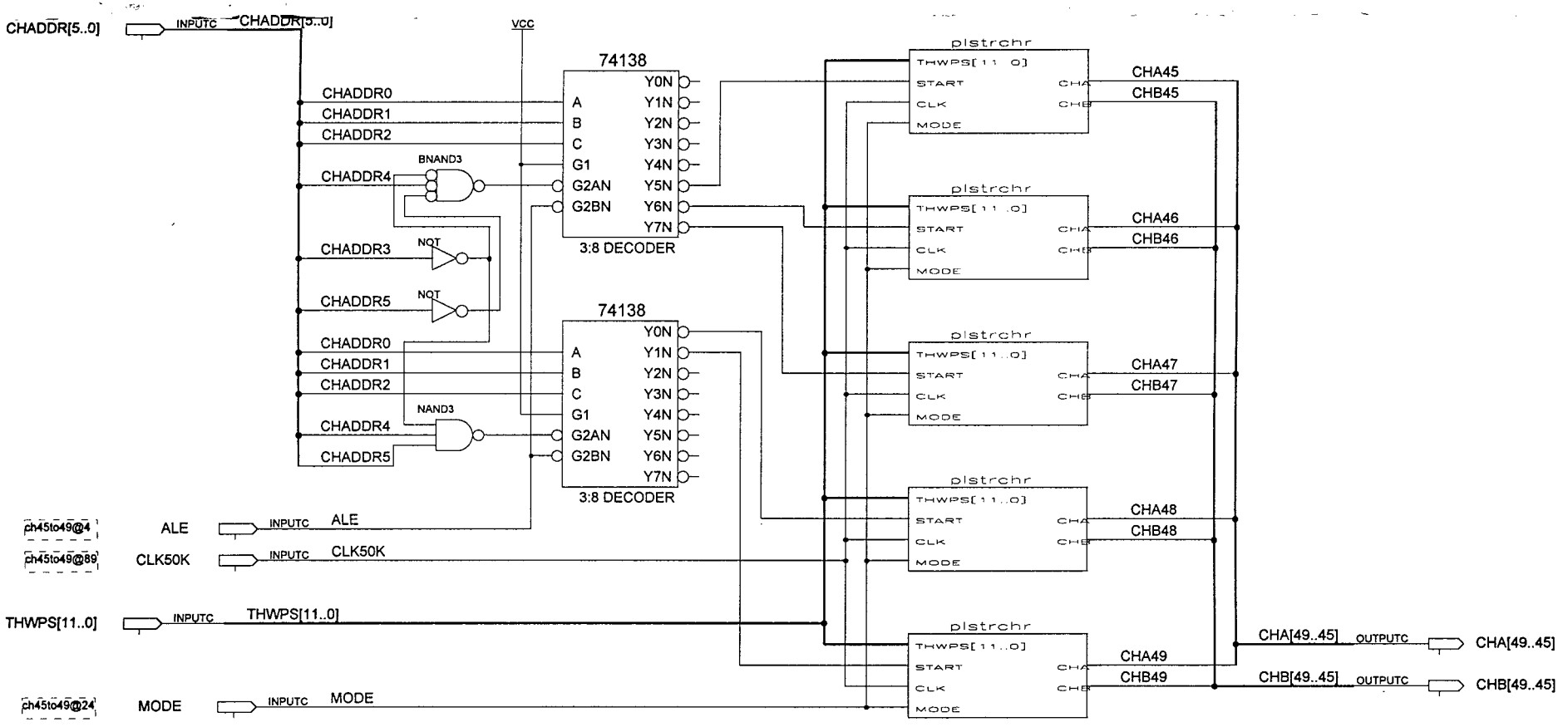
TITLE	LCD DRIVER ELECTRONICS FOR MOE			
COMPANY	Dimensional Media Associates			
DESIGNER	Electronic Design Lab			
SIZE C	NUMBER Channels 30 to 34		REV A	
DATE 10:42a 6-26-1997		SHEET 1	OF 3	



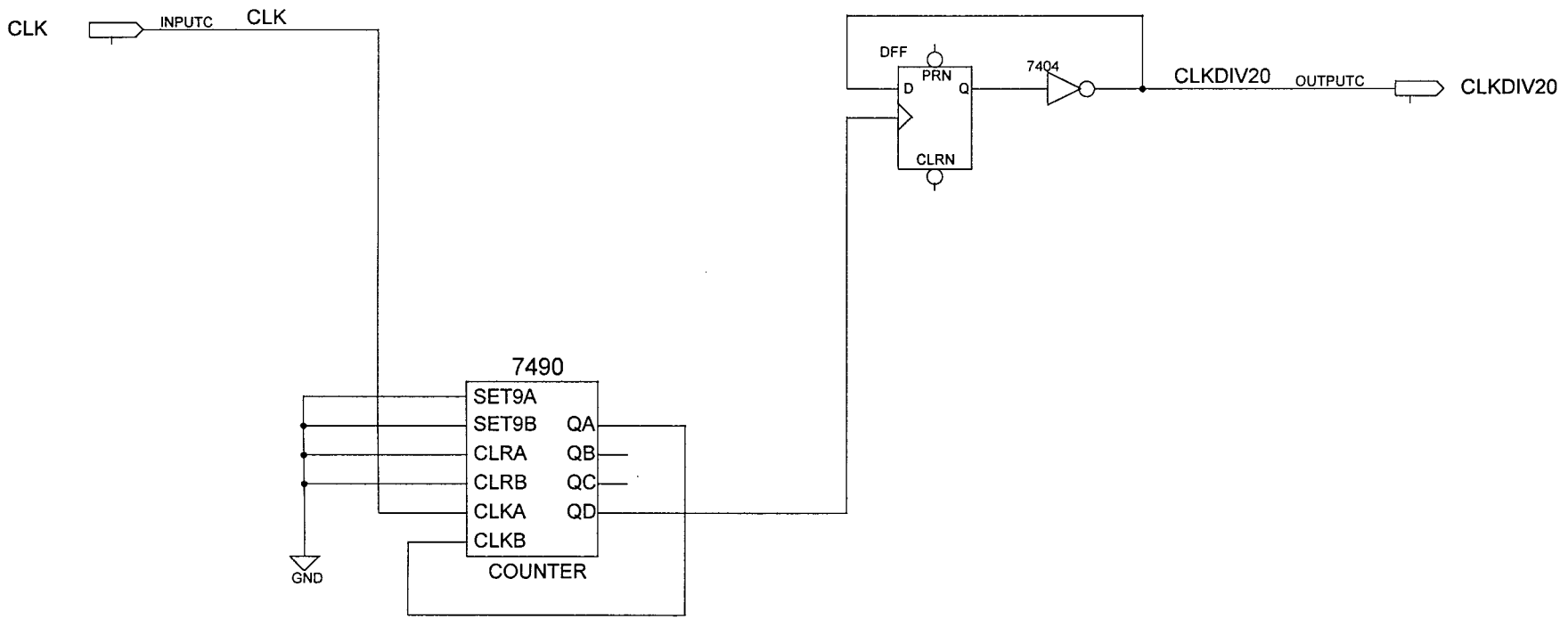
TITLE	LCD DRIVER ELECTRONICS FOR MOE
COMPANY	Dimensional Media Associates
DESIGNER	Electronic Design Lab
SIZE C / NUMBER	Channels 34 to 39 / REV A
DATE	10:58a 6-26-1997
SHEET	1 OF 3



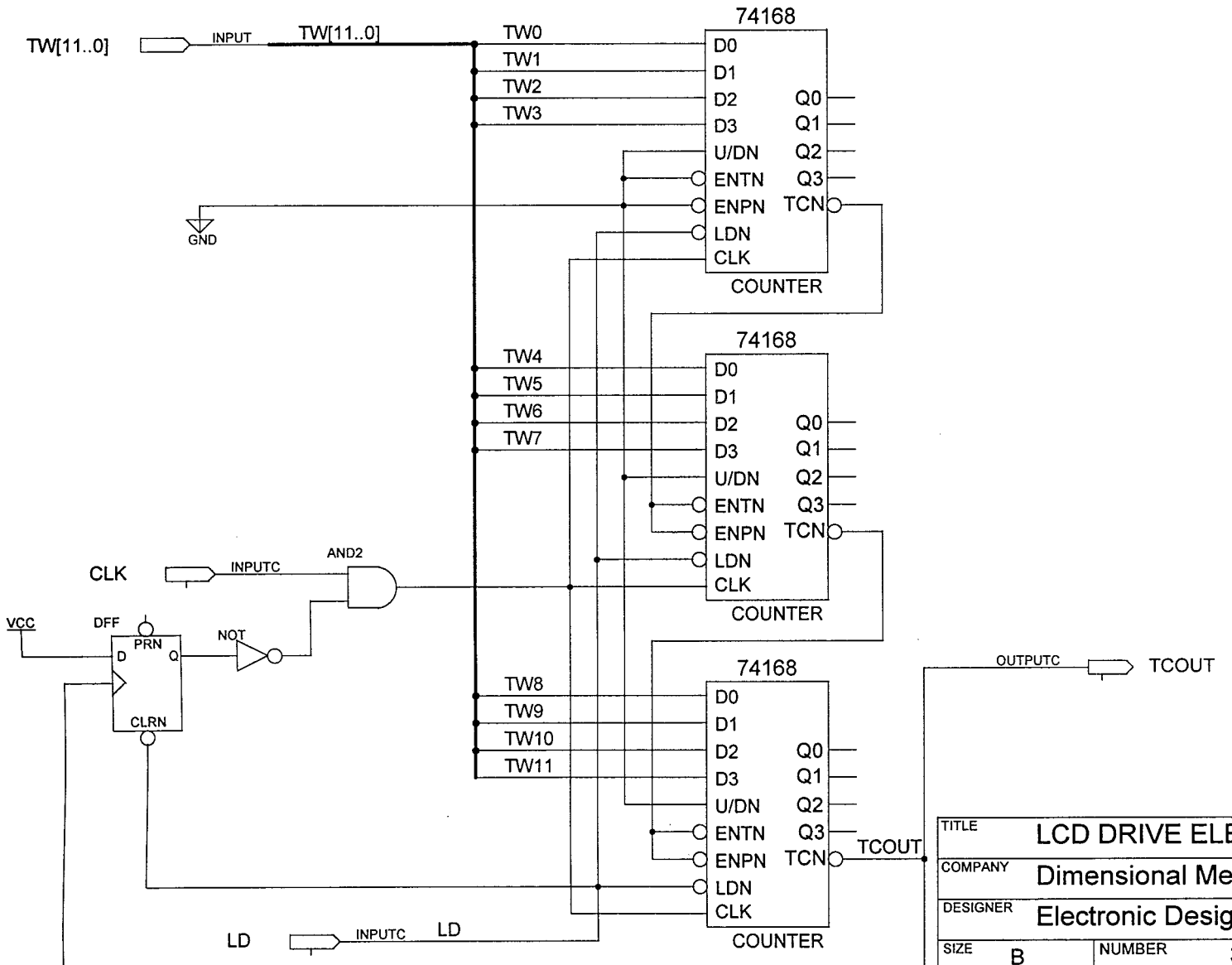
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 40 to 44	REV A	
DATE 11:03a 6-26-1997		SHEET 1	OF 3



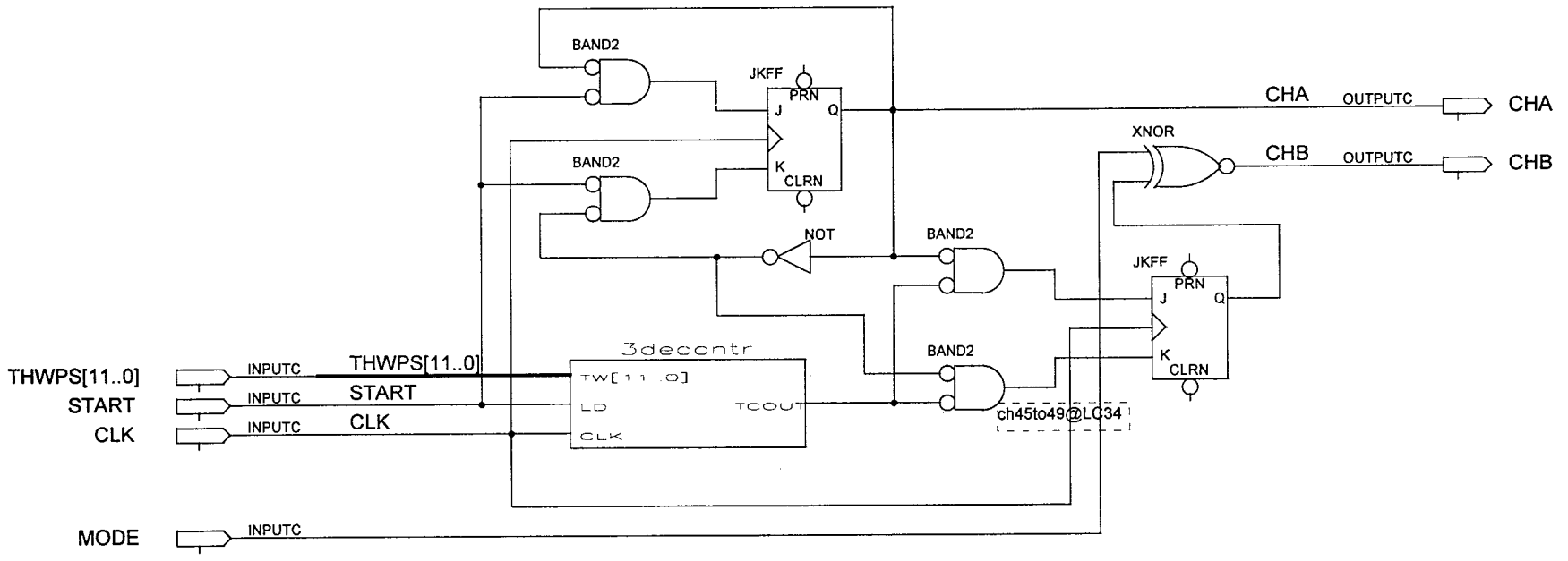
TITLE	LCD DRIVER ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE C	NUMBER Channels 45 to 49	REV A	
DATE 11:08a 6-26-1997	SHEET 1	OF 3	



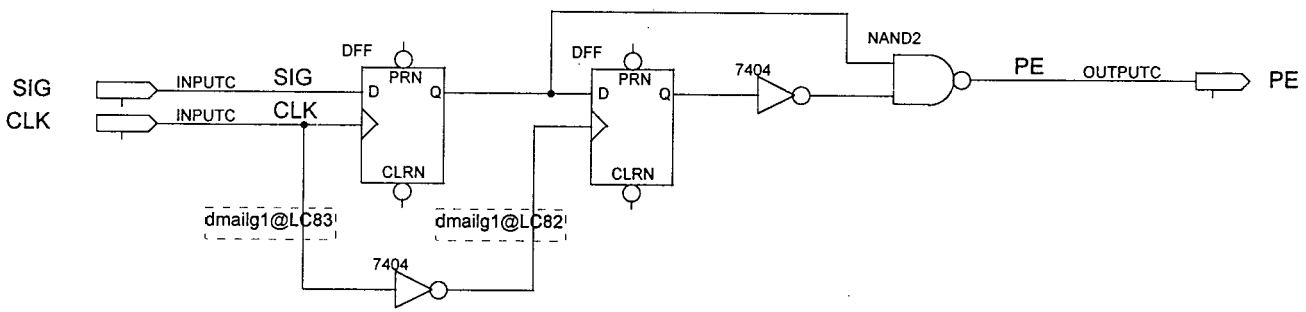
TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE B	NUMBER Divide by 20	REV A	
DATE 10:45p 10-27-1997	SHEET 3	OF 6	



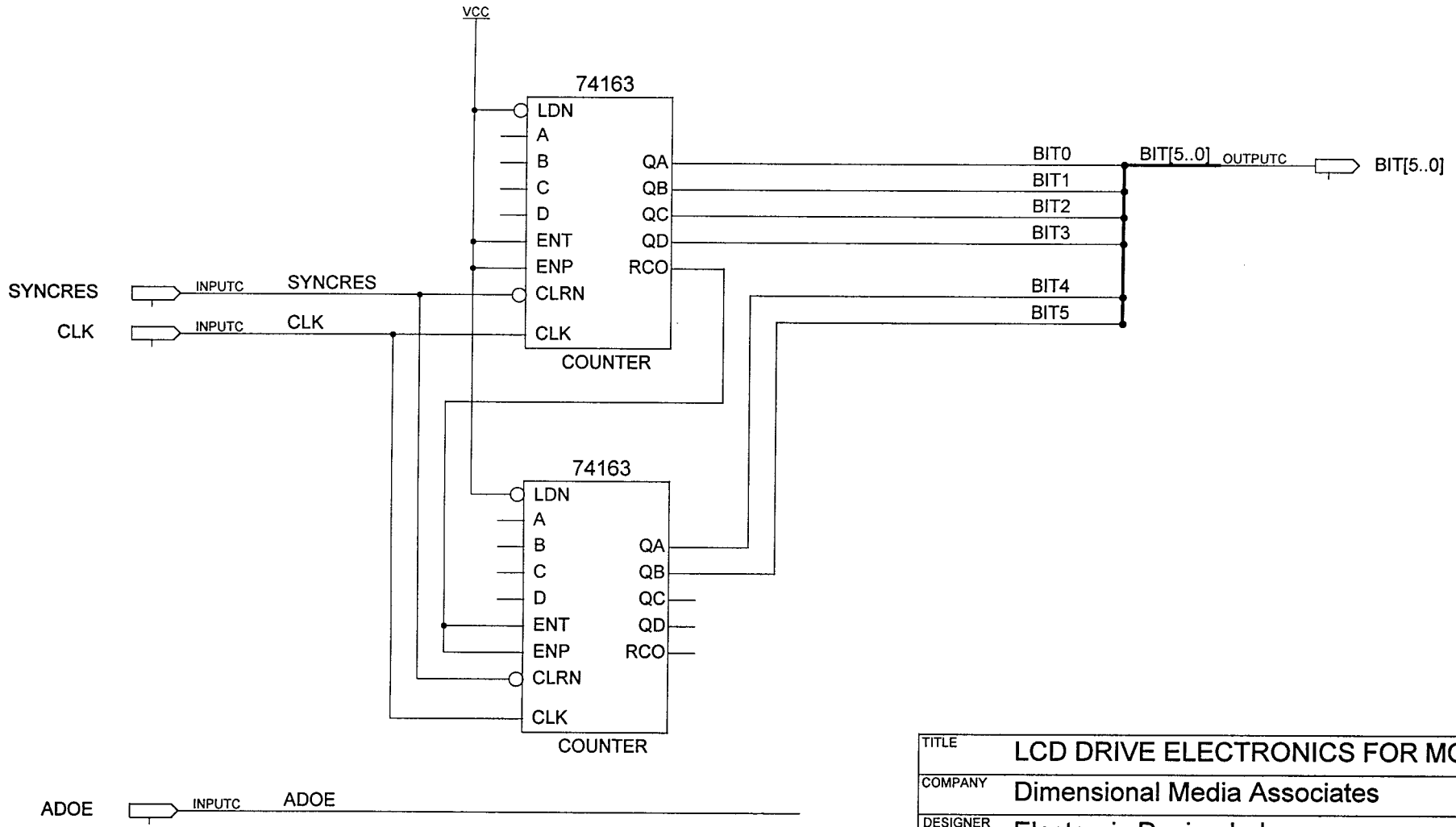
TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE B	NUMBER 3 Decade counter	REV A	
DATE 10:15a 10-30-1997	SHEET 3	OF 3	



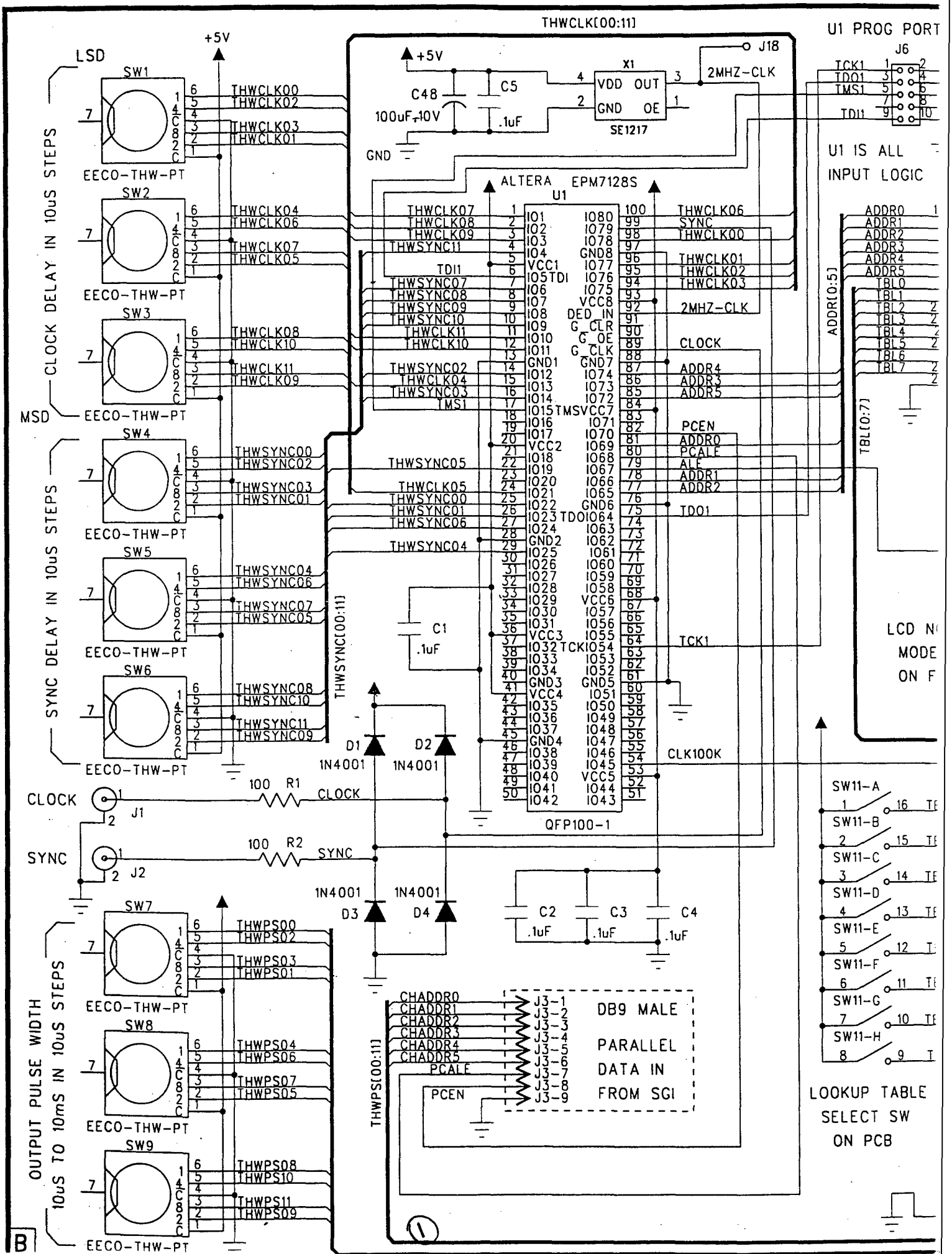
TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE B	NUMBER Output Logic	REV A	
DATE 10:01a 6-26-1997	SHEET 2	OF 3	

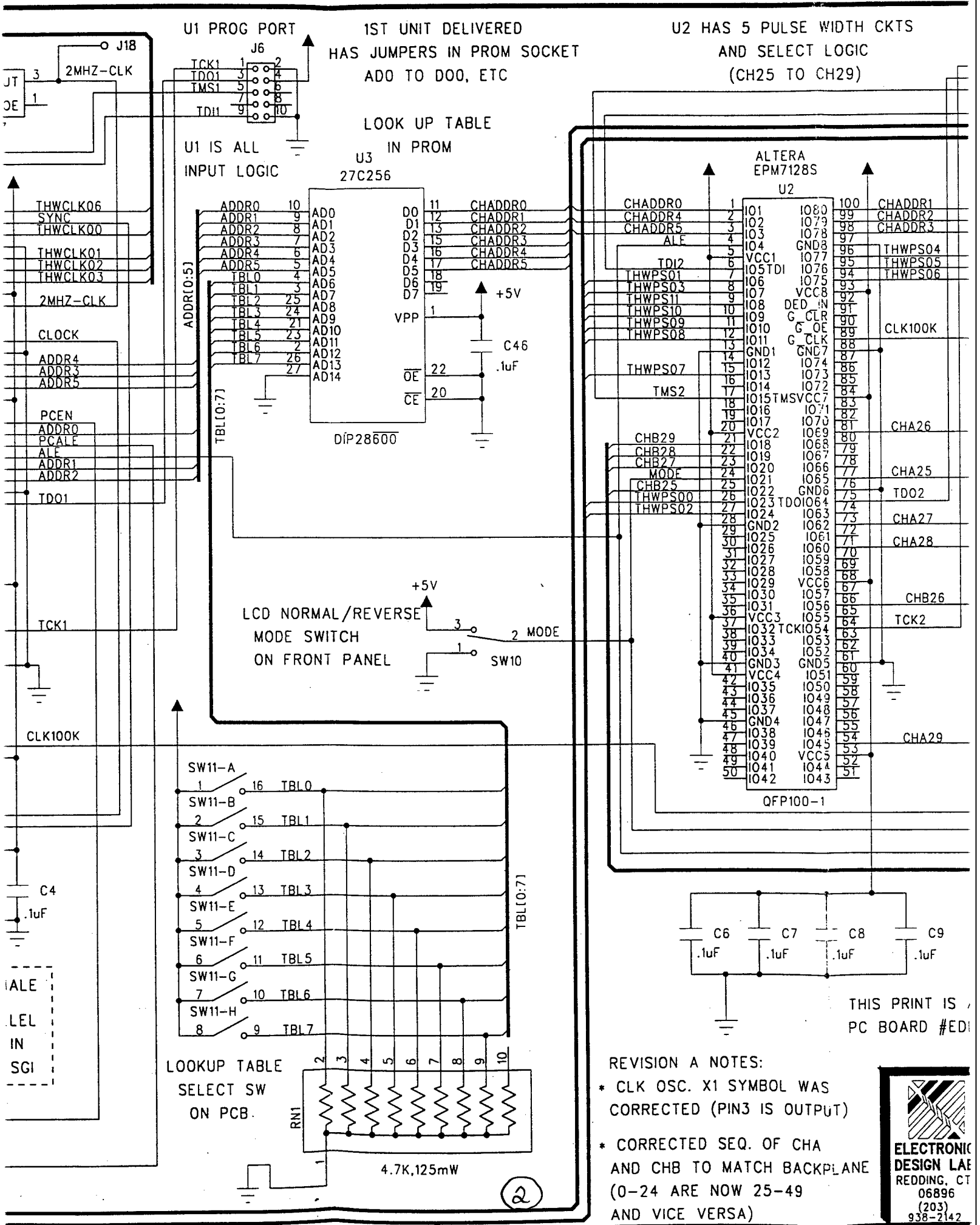


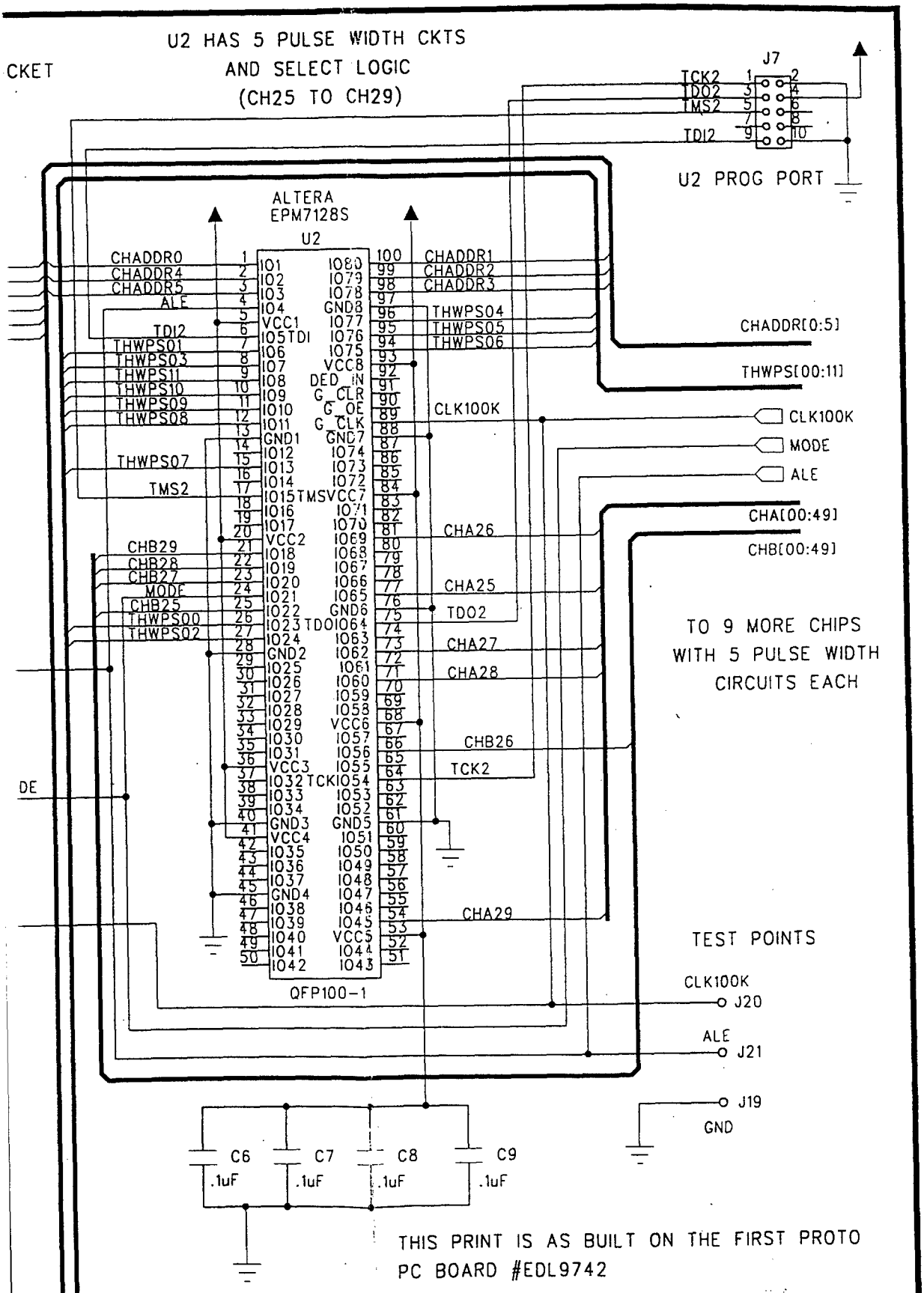
TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronics Design Lab		
SIZE B	NUMBER pos edge detector	REV A	
DATE 3:55p 5-18-2000	SHEET 4 OF 6		



TITLE	LCD DRIVE ELECTRONICS FOR MOE		
COMPANY	Dimensional Media Associates		
DESIGNER	Electronic Design Lab		
SIZE B	NUMBER 6 bit counter	REV A	
DATE 10:05a 10-31-1997	SHEET 6 OF 6		


THWCLK[00:11]
U1 PROG PORT
J6
TCK1
TDO1
TMS1
TDI1
U1 IS ALL
INPUT LOGIC
+5V
LSD
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
EECO-THW-PT
CLOCK DELAY IN 10uS STEPS
SYNC DELAY IN 10uS STEPS
OUTPUT PULSE WIDTH
10uS TO 10mS IN 10uS STEPS
MSD
THWCLK00
THWCLK01
THWCLK02
THWCLK03
THWCLK04
THWCLK05
THWCLK06
THWCLK07
THWCLK08
THWCLK09
THWCLK10
THWCLK11
THWSYNC00
THWSYNC01
THWSYNC02
THWSYNC03
THWSYNC04
THWSYNC05
THWSYNC06
THWSYNC07
THWSYNC08
THWSYNC09
THWSYNC10
THWSYNC11
THWPS00
THWPS01
THWPS02
THWPS03
THWPS04
THWPS05
THWPS06
THWPS07
THWPS08
THWPS09
THWPS10
THWPS11
C48
100uF+10V
C5
.1uF
X1
VDD OUT
GND OE
SE1217
J18
2MHZ-CLK
GND
ALTERA EPM7128S
U1
QFP100-1
TDI1
TMS1
TDO1
TCK1
SYNC
CLOCK
ADDR4
ADDR3
ADDR5
PCEN
ADDR0
PCALE
ALE
ADDR1
ADDR2
CLK100K
ADDR[0:5]
TBL[0:7]
TBL0
TBL1
TBL2
TBL3
TBL4
TBL5
TBL6
TBL7
LCD N
MODE
ON F
C1
.1uF
C2
.1uF
C3
.1uF
C4
.1uF
D1
1N4001
D2
1N4001
D3
1N4001
D4
1N4001
CLOCK
J1
SYNC
J2
100 R1
100 R2
THWSYNC[00:11]
THWPS[00:11]
CHADDR0
CHADDR1
CHADDR2
CHADDR3
CHADDR4
CHADDR5
PCALE
PCEN
J3-1
J3-2
J3-3
J3-4
J3-5
J3-6
J3-7
J3-8
J3-9
DB9 MALE
PARALLEL
DATA IN
FROM SGI
SW11-A
SW11-B
SW11-C
SW11-D
SW11-E
SW11-F
SW11-G
SW11-H
LOOKUP TABLE
SELECT SW
ON PCB
B


U1 PROG PORT
1ST UNIT DELIVERED
HAS JUMPERS IN PROM SOCKET
AD0 TO D00, ETC
U2 HAS 5 PULSE WIDTH CKTS
AND SELECT LOGIC
(CH25 TO CH29)
LOOK UP TABLE
IN PROM
U1 IS ALL
INPUT LOGIC
U3
27C256
ALTERA
EPM7128S
U2
J18
J6
2MHZ-CLK
TCK1
TDO1
TMS1
TDI1
THWCLK06
SYNC
THWCLK00
THWCLK01
THWCLK02
THWCLK03
2MHZ-CLK
CLOCK
ADDR4
ADDR3
ADDR5
PCEN
ADDR0
PCALE
ALE
ADDR1
ADDR2
TDO1
TCK1
CLK100K
ADDR0:5
TBL0:7
ADDR0
ADDR1
ADDR2
ADDR3
ADDR4
ADDR5
TBL0
TBL1
TBL2
TBL3
TBL4
TBL5
TBL6
TBL7
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
AD8
AD9
AD10
AD11
AD12
AD13
AD14
D0
D1
D2
D3
D4
D5
D6
D7
VPP
OE
CE
CHADDR0
CHADDR1
CHADDR2
CHADDR3
CHADDR4
CHADDR5
+5V
C46
.1uF
DIP28600
CHADDR0
CHADDR4
CHADDR5
ALE
TDI2
THWPS01
THWPS03
THWPS11
THWPS10
THWPS09
THWPS08
THWPS07
TMS2
CHB29
CHB28
CHB27
MODE
CHB25
THWPS00
THWPS02
GND1
VCC2
VCC3
TCK1
GND2
GND3
VCC4
GND4
VCC5
GND5
VCC6
GND6
TDO1
GND7
VCC7
GND8
VCC8
DED_IN
G_CLR
G_OE
G_CLK
CLK100K
CHA26
CHA25
TDO2
CHA27
CHA28
CHB26
TCK2
CHA29
QFP100-1
LCD NORMAL/REVERSE
MODE SWITCH
ON FRONT PANEL
+5V
MODE
SW10
SW11-A
SW11-B
SW11-C
SW11-D
SW11-E
SW11-F
SW11-G
SW11-H
TBL0
TBL1
TBL2
TBL3
TBL4
TBL5
TBL6
TBL7
TBL0:7
LOOKUP TABLE
SELECT SW
ON PCB.
RN1
4.7K,125mW
C4
.1uF
C6
C7
C8
C9
.1uF
THIS PRINT IS
PC BOARD #ED
REVISION A NOTES:
* CLK OSC. X1 SYMBOL WAS CORRECTED (PIN3 IS OUTPUT)
* CORRECTED SEQ. OF CHA AND CHB TO MATCH BACKPLANE (0-24 ARE NOW 25-49 AND VICE VERSA)
ELECTRONIC
DESIGN LAB
REDDING, CT
06896
(203)
938-2142
2



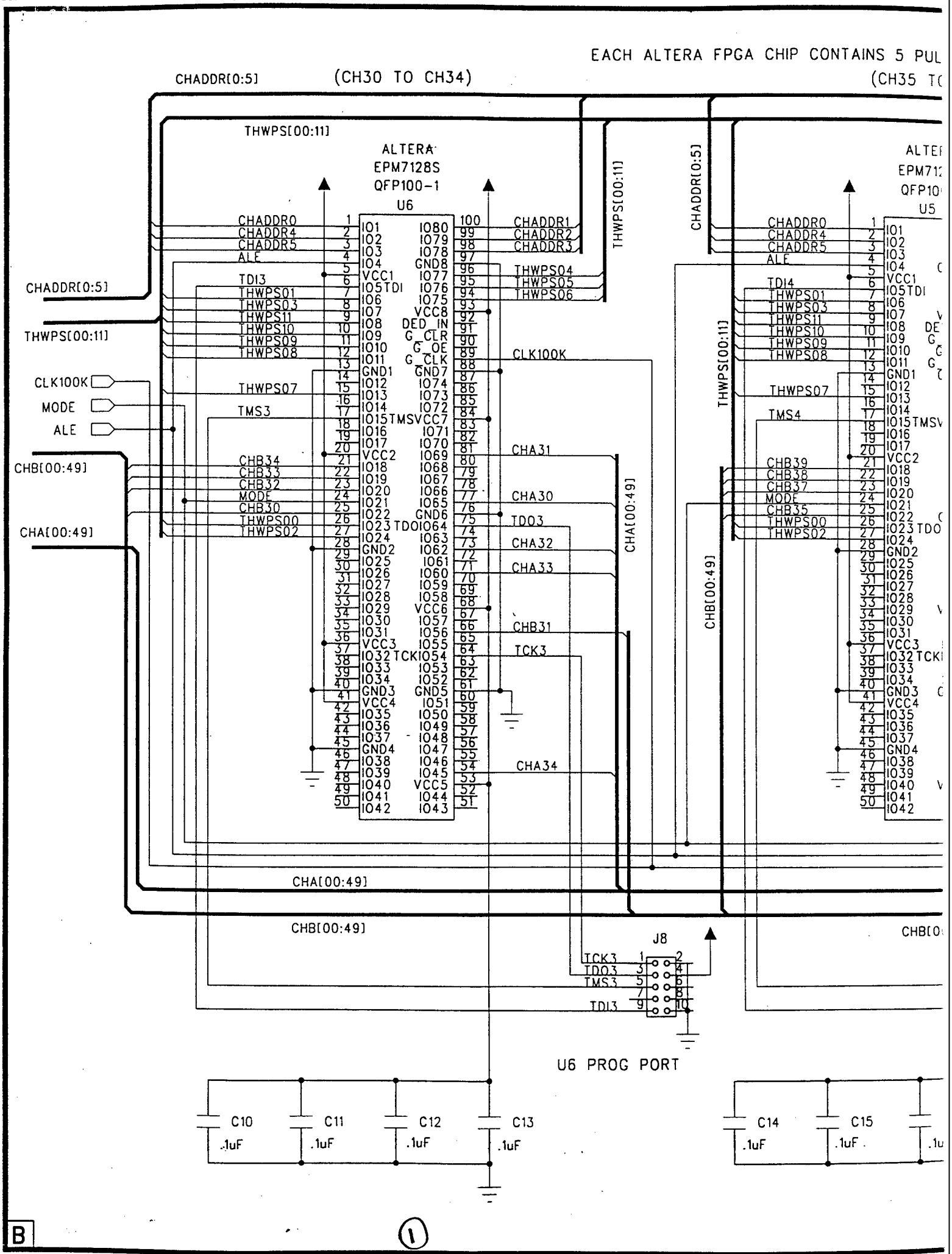
REVISION A NOTES:

* CLK OSC. X1 SYMBOL WAS CORRECTED (PIN3 IS OUTPUT)
* CORRECTED SEQ. OF CHA AND CHB TO MATCH BACKPLANE (0-24 ARE NOW 25-49 AND VICE VERSA)

(3)

ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203) 938-2142

CLIENT	DIMENSIONAL MEDIA ASSOC		
PROJECT	MOE DRIVER		REV A
LOGIC BOARD INPUT LOGIC AND PULSE STRETCHER CHANNELS 0 TO 4			
DRAWING BY	JEP	DATE	7-1-97
DESIGN BY	JOG		
JOB 601	DRAWING#	#: 1 OF 5	

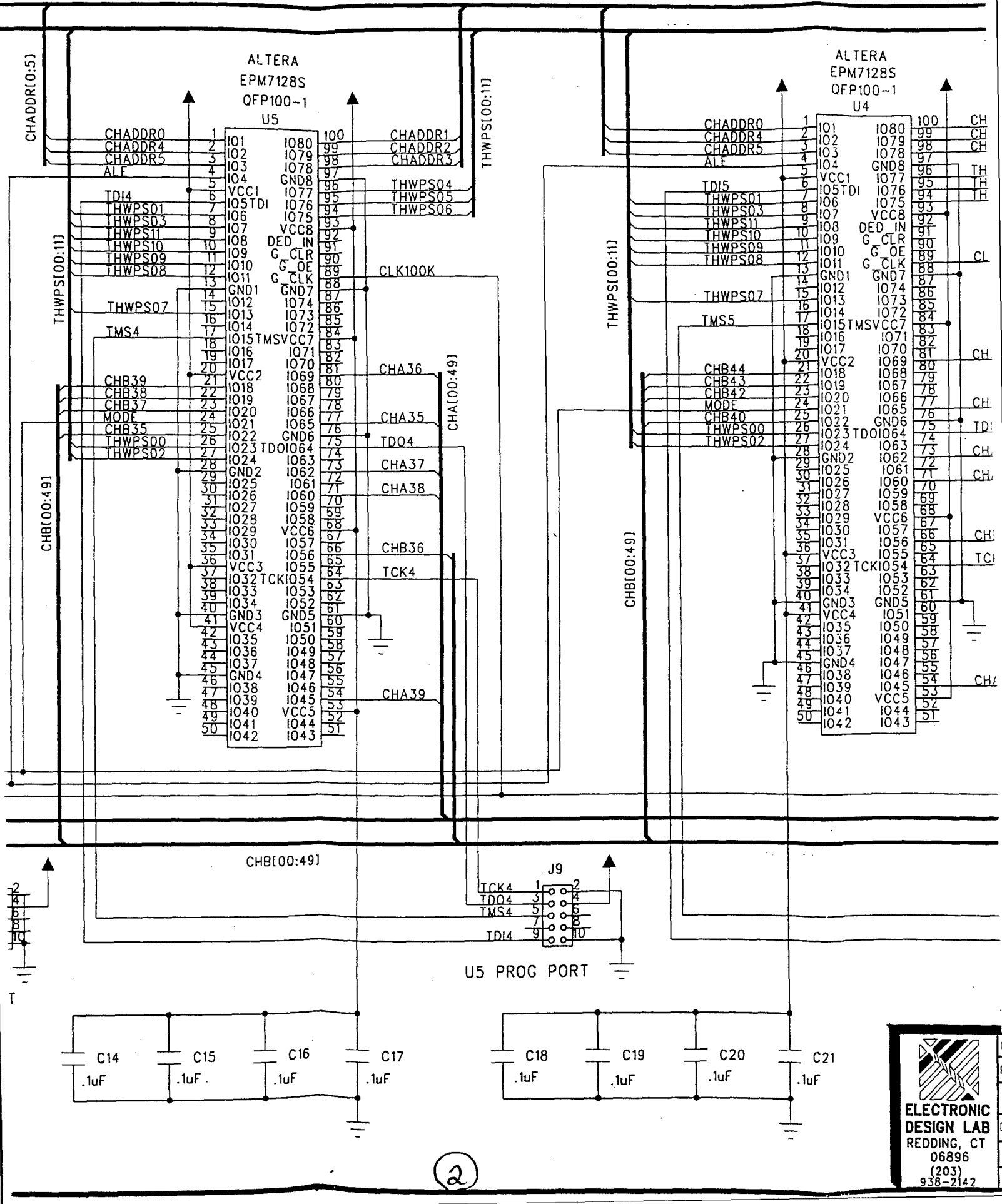

EACH ALTERA FPGA CHIP CONTAINS 5 PUL
CHADDR[0:5]
(CH30 TO CH34)
(CH35 TO
THWPS[00:11]
ALTERA
EPM7128S
QFP100-1
U6
CHADDR0
CHADDR4
CHADDR5
ALE
TDI3
THWPS01
THWPS03
THWPS11
THWPS10
THWPS09
THWPS08
THWPS07
TMS3
CHB34
CHB33
CHB32
MODE
CHB30
THWPS00
THWPS02
CHADDR1
CHADDR2
CHADDR3
THWPS04
THWPS05
THWPS06
CLK100K
CHA31
CHA30
TDO3
CHA32
CHA33
CHB31
TCK3
CHA34
CLK100K
MODE
ALE
CHB[00:49]
CHA[00:49]
U5
TDI4
TMS4
CHB39
CHB38
CHB37
CHB35
J8
TCK3
TDO3
TMS3
TDI3
U6 PROG PORT
C10
C11
C12
C13
C14
C15
.1uF
B
1

.TERA FPGA CHIP CONTAINS 5 PULSE WIDTH CKTS AND SELECT LOGIC

(CH35 TO CH39)

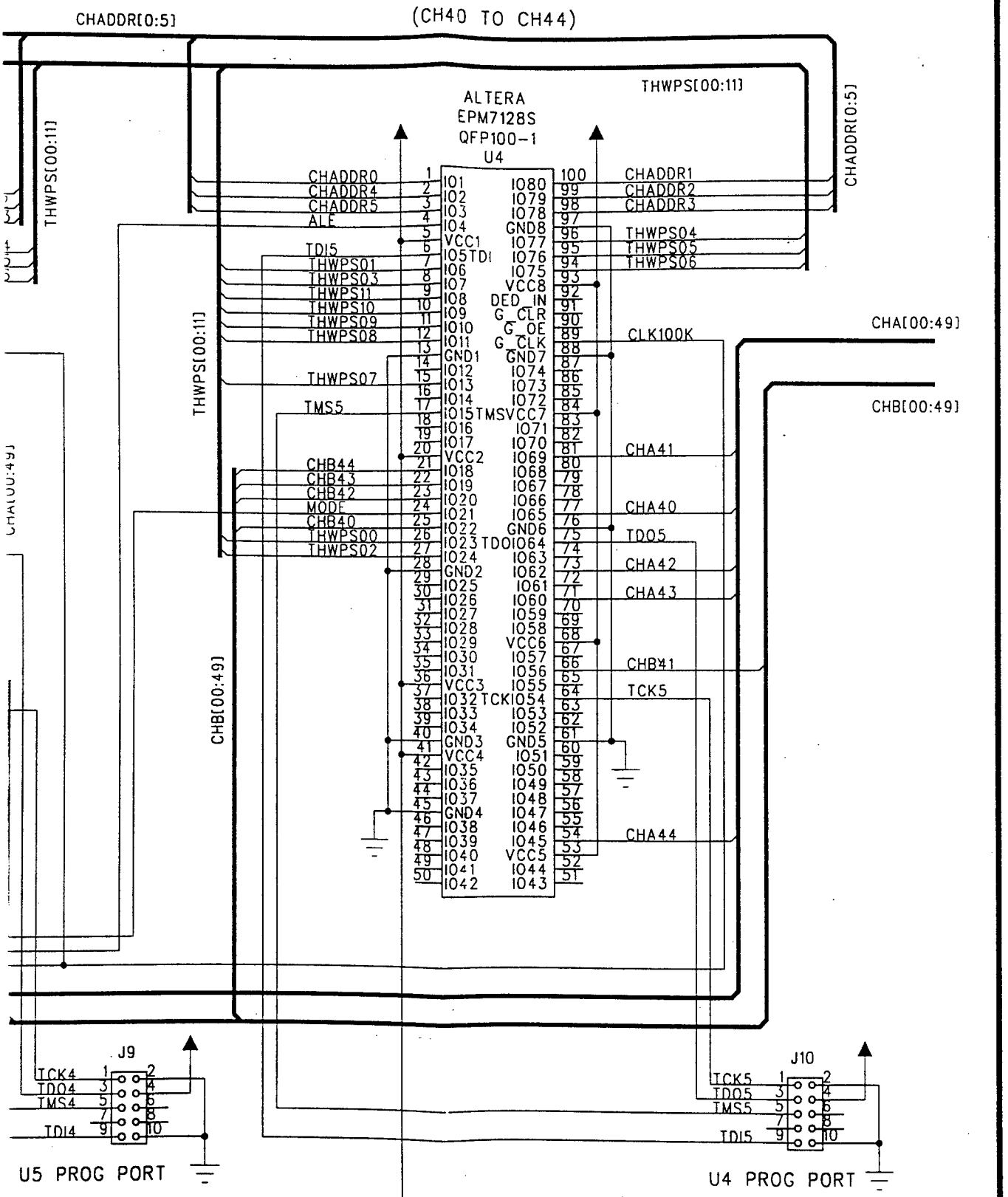
CHADDR[0:5]

(CH40 TO CH44)



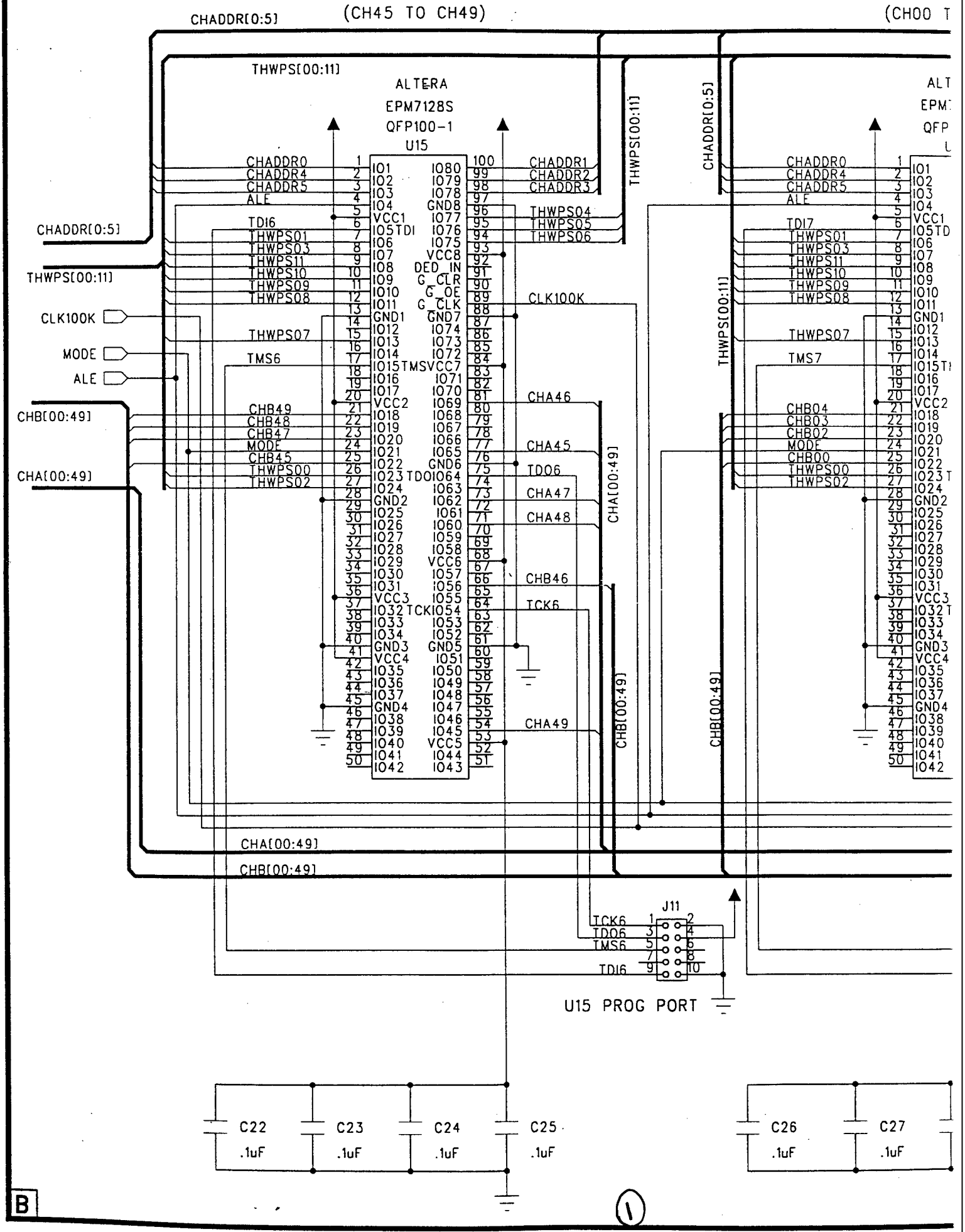
ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203) 938-2142

D SELECT LOGIC

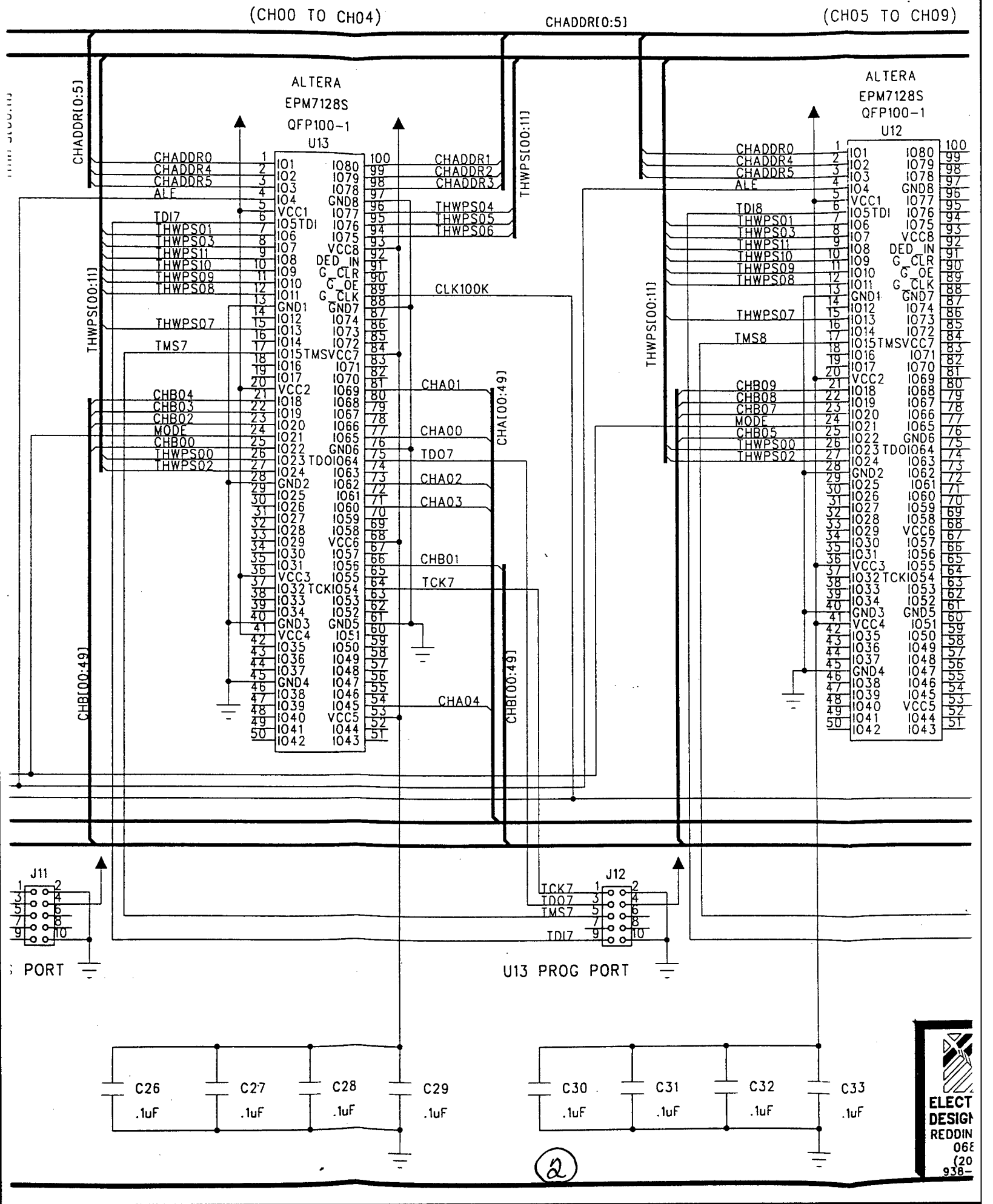


ELECTRONIC DESIGN LAB REDDING, CT 06896 (203) 938-2142	CLIENT	DIMENSIONAL MEDIA ASSOC	
	PROJECT	MOE DRIVER	REV A
	LOGIC BOARD PULSE STRETCHER CHANNELS 5 TO 19		
	DRAWING BY	JEP	DATE 7-1-97
	DESIGN BY	JOG	
	JOB 601	DRAWING#	#: 2 OF 5

EACH ALTERA FPGA CHIP CONTAINS 5

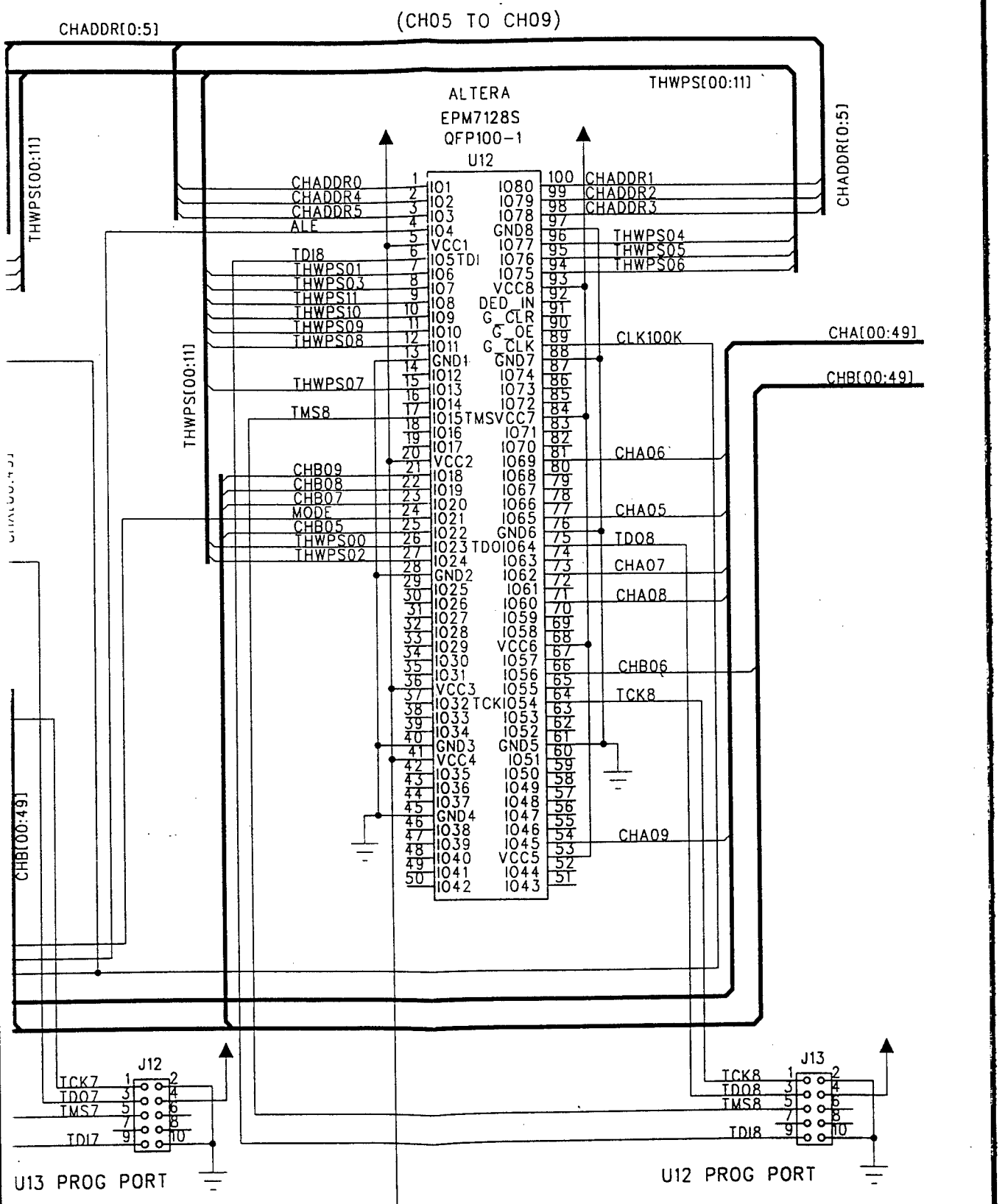


EACH ALTERA FPGA CHIP CONTAINS 5 PULSE WIDTH CKTS AND SELECT LOGIC



ND SELECT LOGIC

(CH05 TO CH09)



C30 .1uF

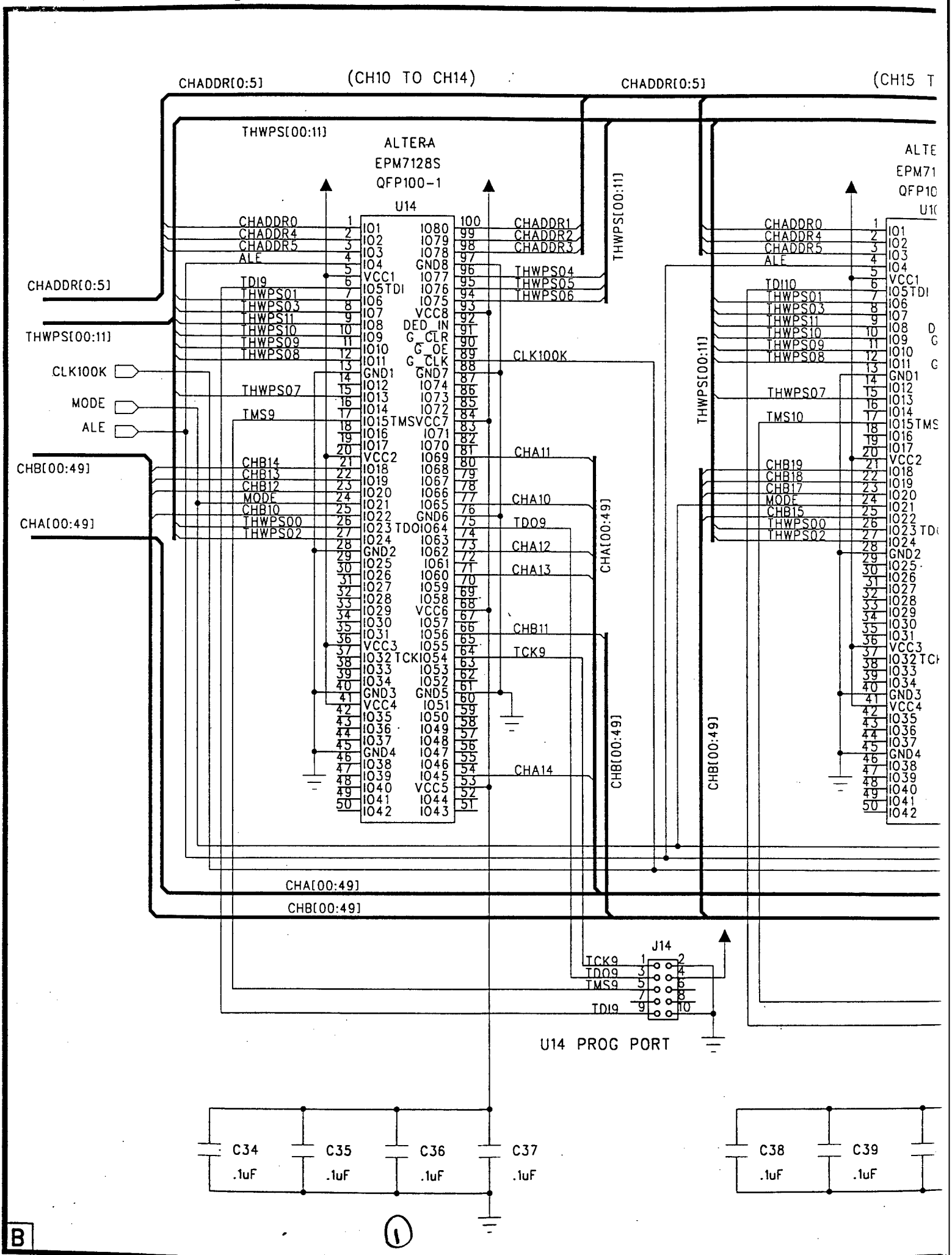
C31 .1uF

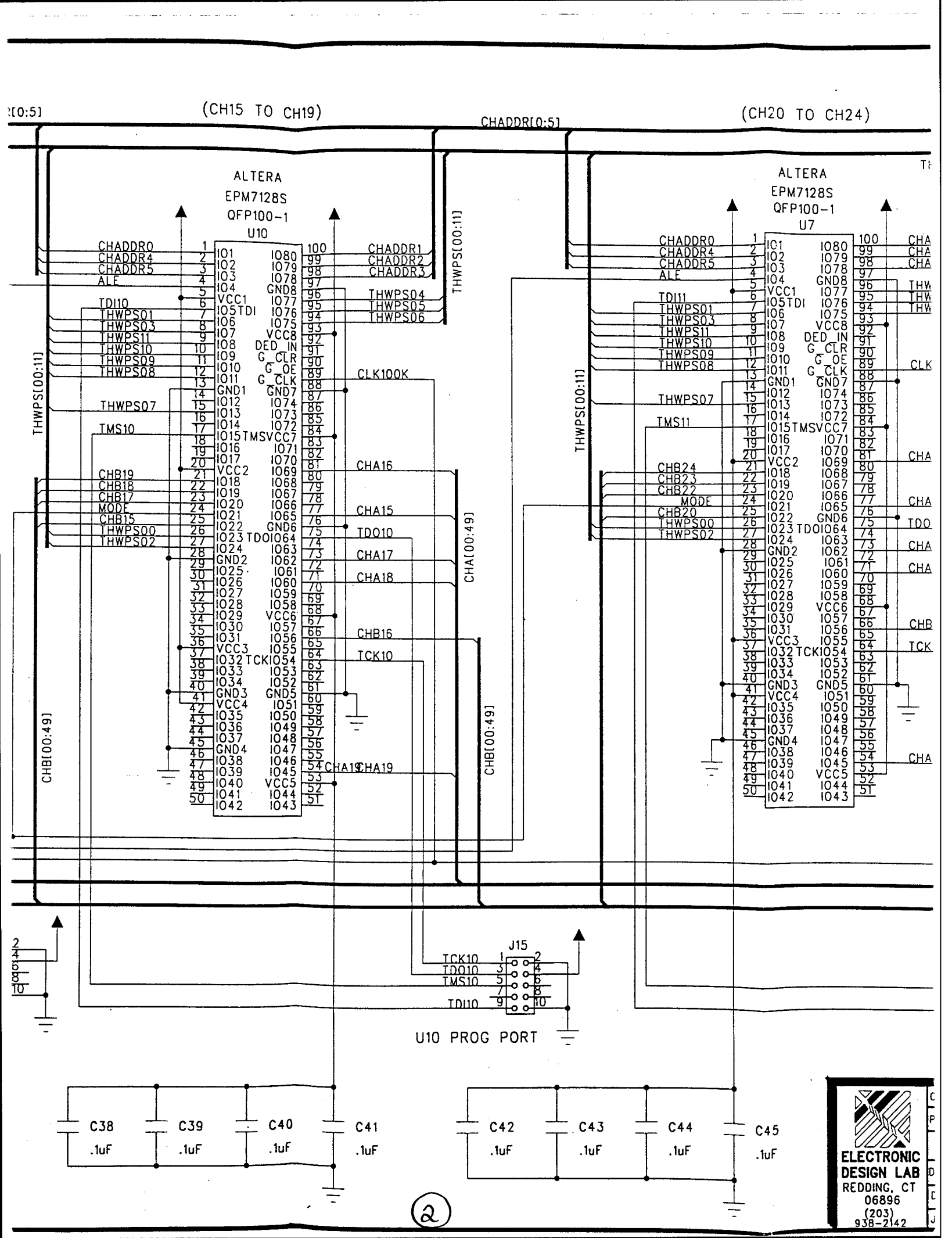
C32 .1uF

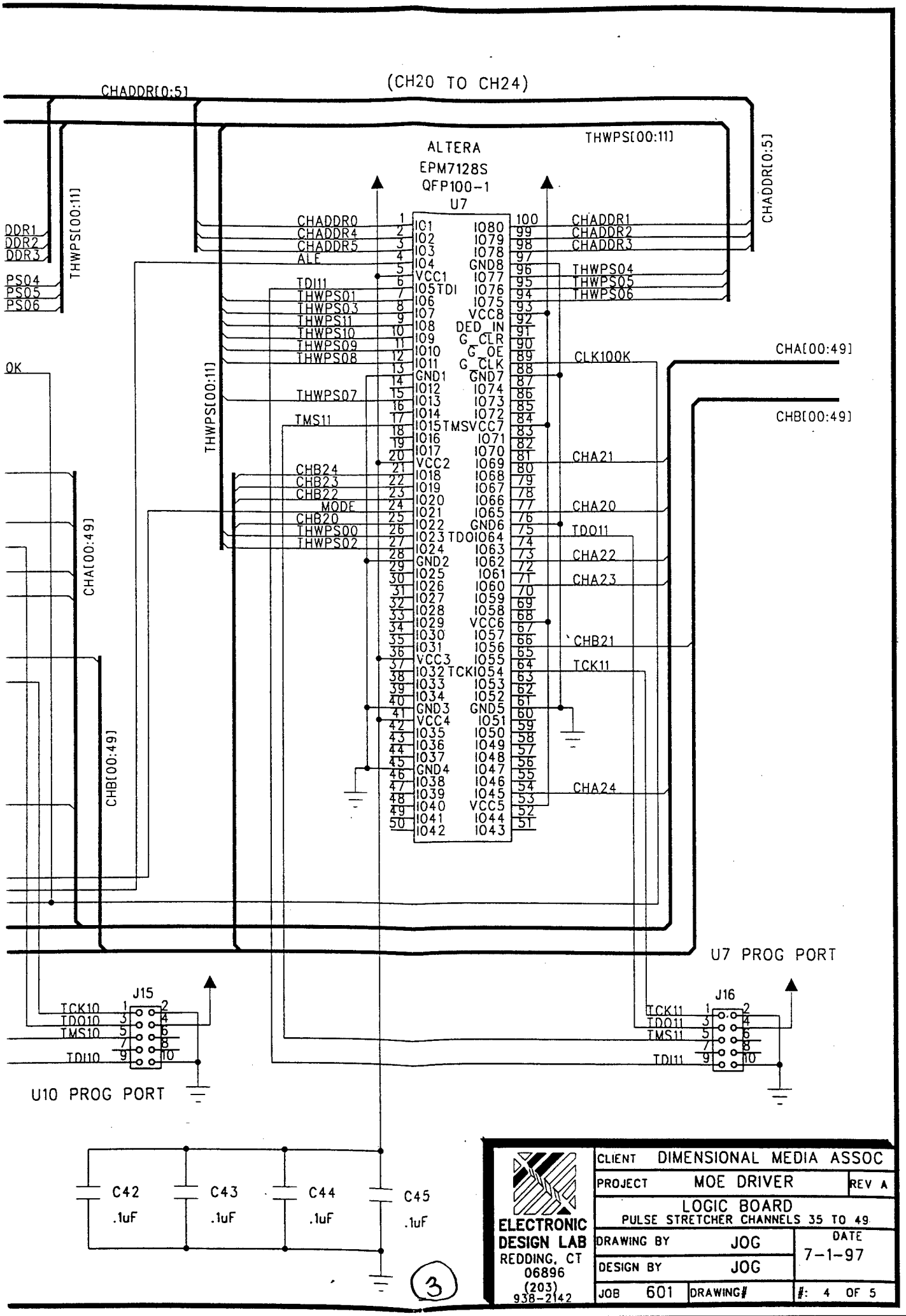
C33 .1uF

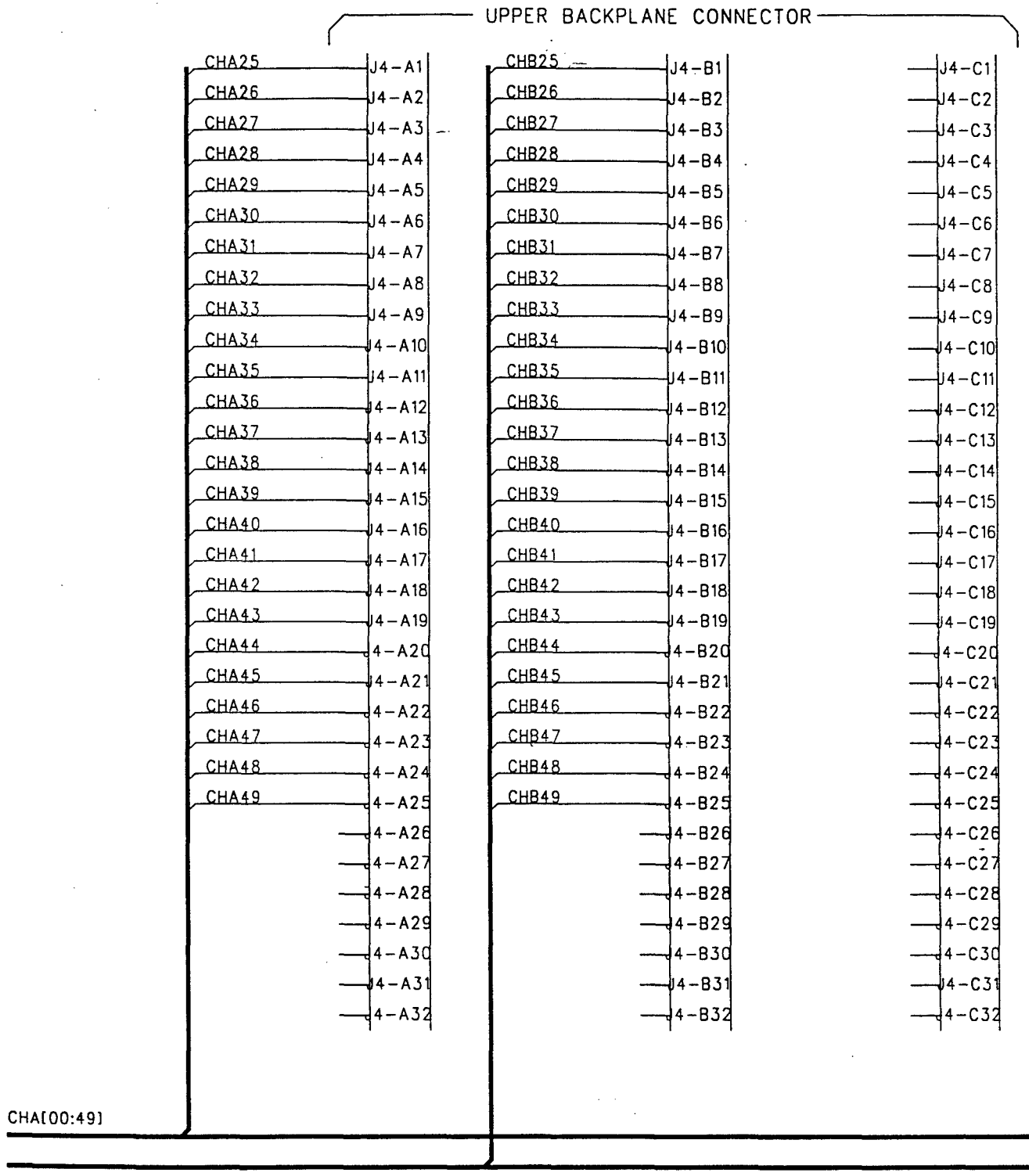
ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203) 938-2142

CLIENT	DIMENSIONAL MEDIA ASSOC		
PROJECT	MOE DRIVER		REV A
LOGIC BOARD PULSE STRETCHER CHANNELS 20 TO 34			
DRAWING BY	JOG	DATE	7-1-97
DESIGN BY	JOG		
JOB	601	DRAWING#	#: 3 OF 5


CHADDR[0:5]
(CH10 TO CH14)
CHADDR[0:5]
(CH15 T
THWPS[00:11]
ALTERA
EPM7128S
QFP100-1
U14
CHADDR0
CHADDR4
CHADDR5
ALE
TDI9
THWPS01
THWPS03
THWPS11
THWPS10
THWPS09
THWPS08
THWPS07
TMS9
CHB14
CHB13
CHB12
MODE
CHB10
THWPS00
THWPS02
CHADDR1
CHADDR2
CHADDR3
THWPS04
THWPS05
THWPS06
CLK100K
CHA11
CHA10
TDO9
CHA12
CHA13
CHB11
TCK9
CHA14
CHA[00:49]
CHB[00:49]
CLK100K
MODE
ALE
J14
TCK9
TDO9
TMS9
TDI9
U14 PROG PORT
C34
.1uF
C35
.1uF
C36
.1uF
C37
.1uF
C38
.1uF
C39
.1uF
B

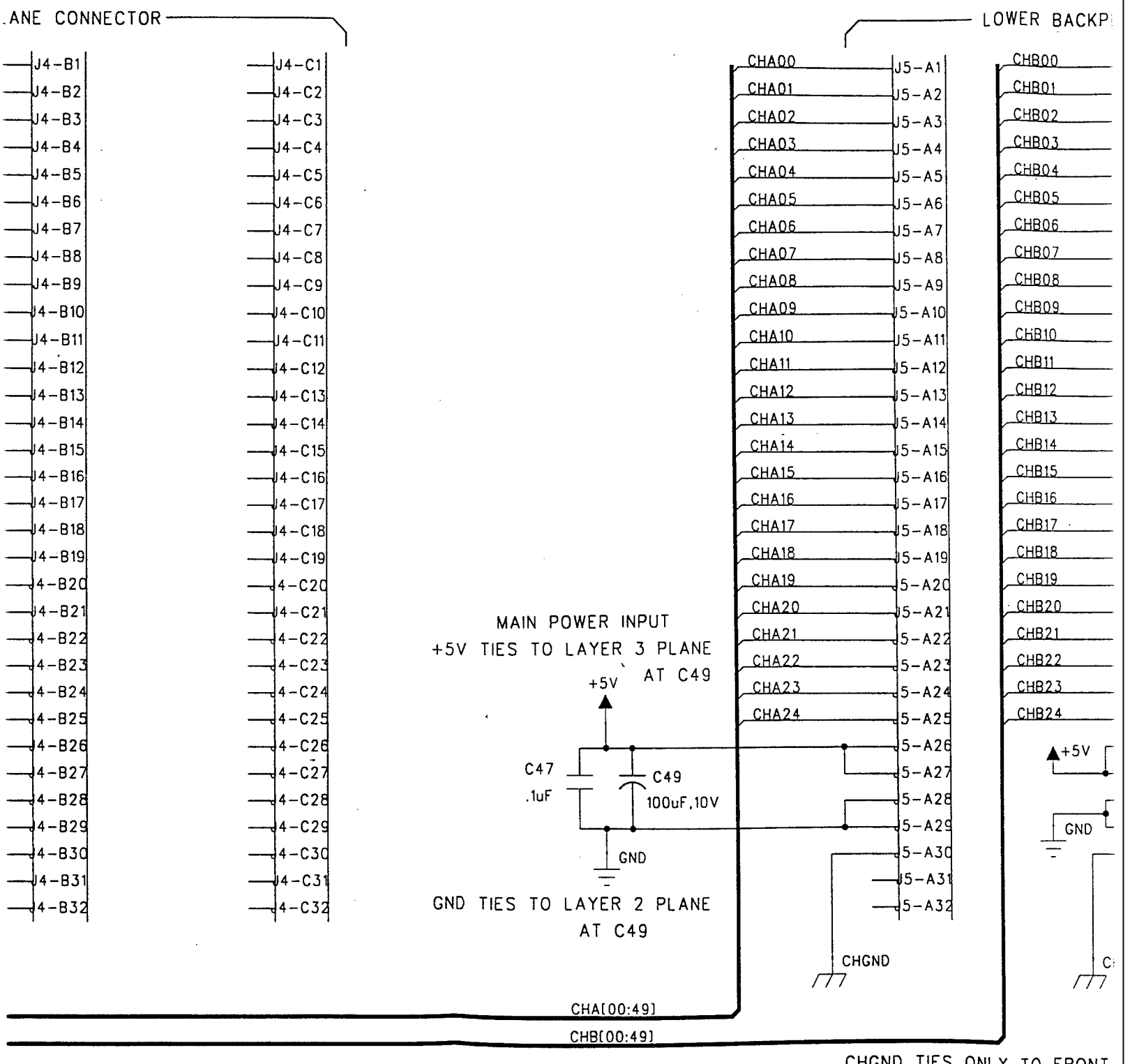

(CH15 TO CH19)
CHADDR[0:5]
(CH20 TO CH24)
ALTERA
EPM7128S
QFP100-1
U10
U7
THWPS[00:11]
CHA[00:49]
CHB[00:49]
CLK100K
TMS10
TMS11
TDI10
TDI11
TDO10
TCK10
CHA16
CHA15
CHA17
CHA18
CHB16
CHA19
J15
U10 PROG PORT
C38 .1uF
C39 .1uF
C40 .1uF
C41 .1uF
C42 .1uF
C43 .1uF
C44 .1uF
C45 .1uF
ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203)
938-2142
2


(CH20 TO CH24)
CHADDR[0:5]
THWPS[00:11]
ALTERA
EPM7128S
QFP100-1
U7
CHADDR0
CHADDR4
CHADDR5
ALE
TDI11
THWPS01
THWPS03
THWPS11
THWPS10
THWPS09
THWPS08
THWPS07
TMS11
CHB24
CHB23
CHB22
MODE
CHB20
THWPS00
THWPS02
CHADDR1
CHADDR2
CHADDR3
THWPS04
THWPS05
THWPS06
CLK100K
CHA21
CHA20
TDO11
CHA22
CHA23
CHB21
TCK11
CHA24
DDR1
DDR2
DDR3
PS04
PS05
PS06
OK
CHA[00:49]
CHB[00:49]
U7 PROG PORT
J15
J16
TCK10
TDO10
TMS10
TDI10
TCK11
TDO11
TMS11
TDI11
U10 PROG PORT
C42
.1uF
C43
.1uF
C44
.1uF
C45
.1uF
3
ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203)
938-2142
CLIENT DIMENSIONAL MEDIA ASSOC
PROJECT MOE DRIVER
REV A
LOGIC BOARD
PULSE STRETCHER CHANNELS 35 TO 49
DRAWING BY JOG
DESIGN BY JOG
DATE 7-1-97
JOB 601
DRAWING#
#: 4 OF 5

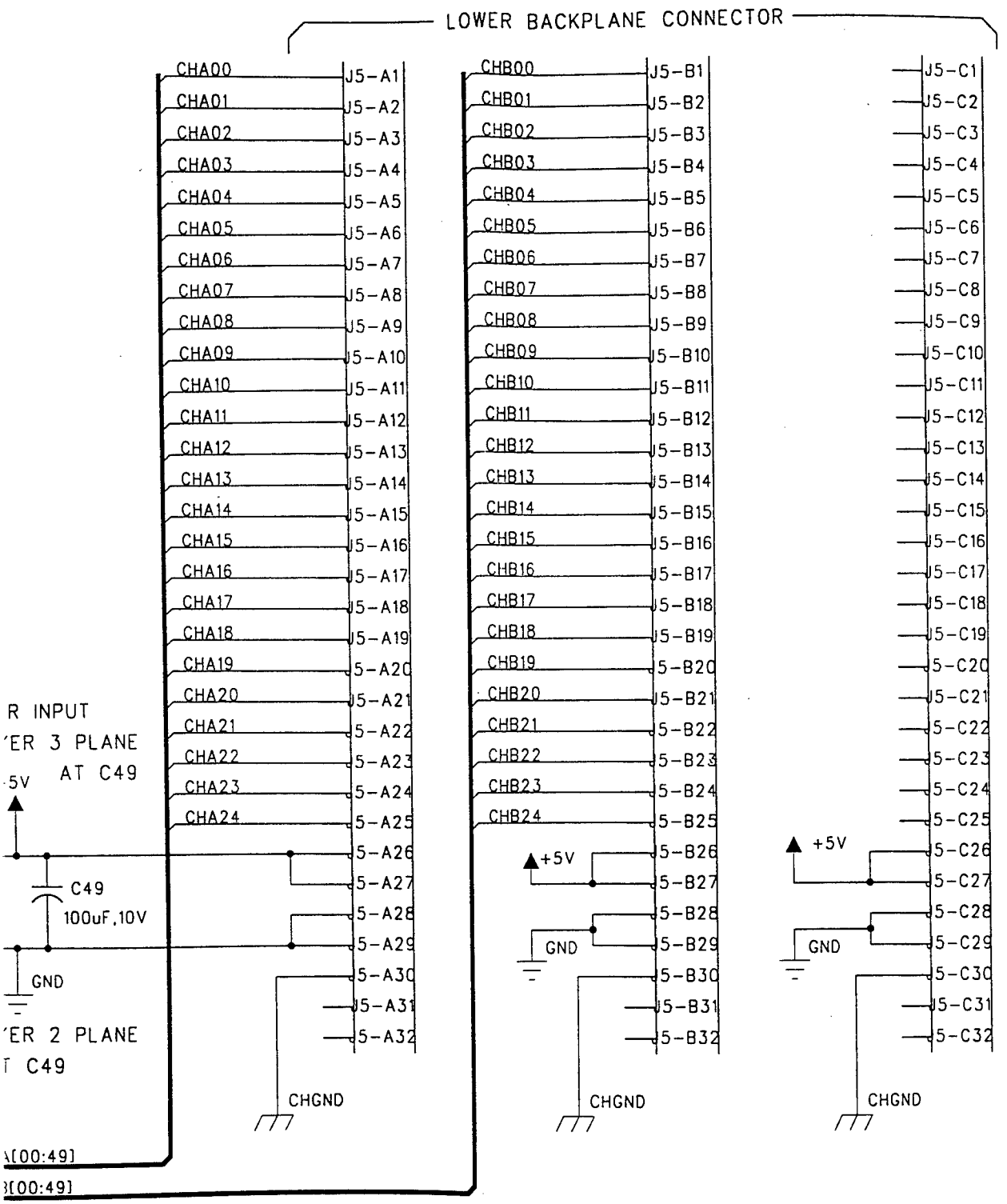


96 PIN MALE DIN CONNECTORS TO INTER

B



96 PIN MALE DIN CONNECTORS TO INTERFACE WITH VME SIZE 6U BACKPLANE

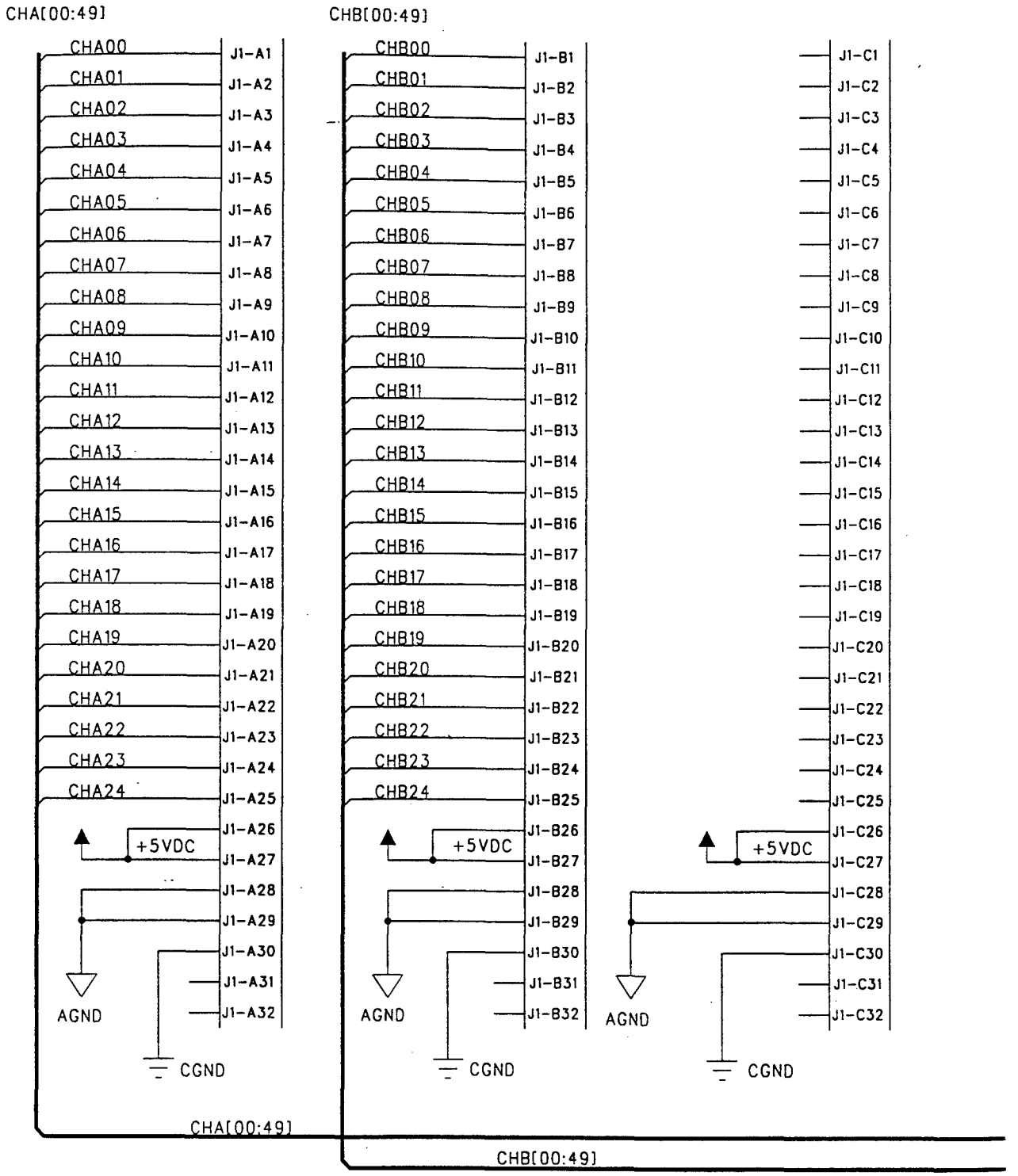


CHGND TIES ONLY TO FRONT PANEL BRACKET

E 6U BACKPLANE

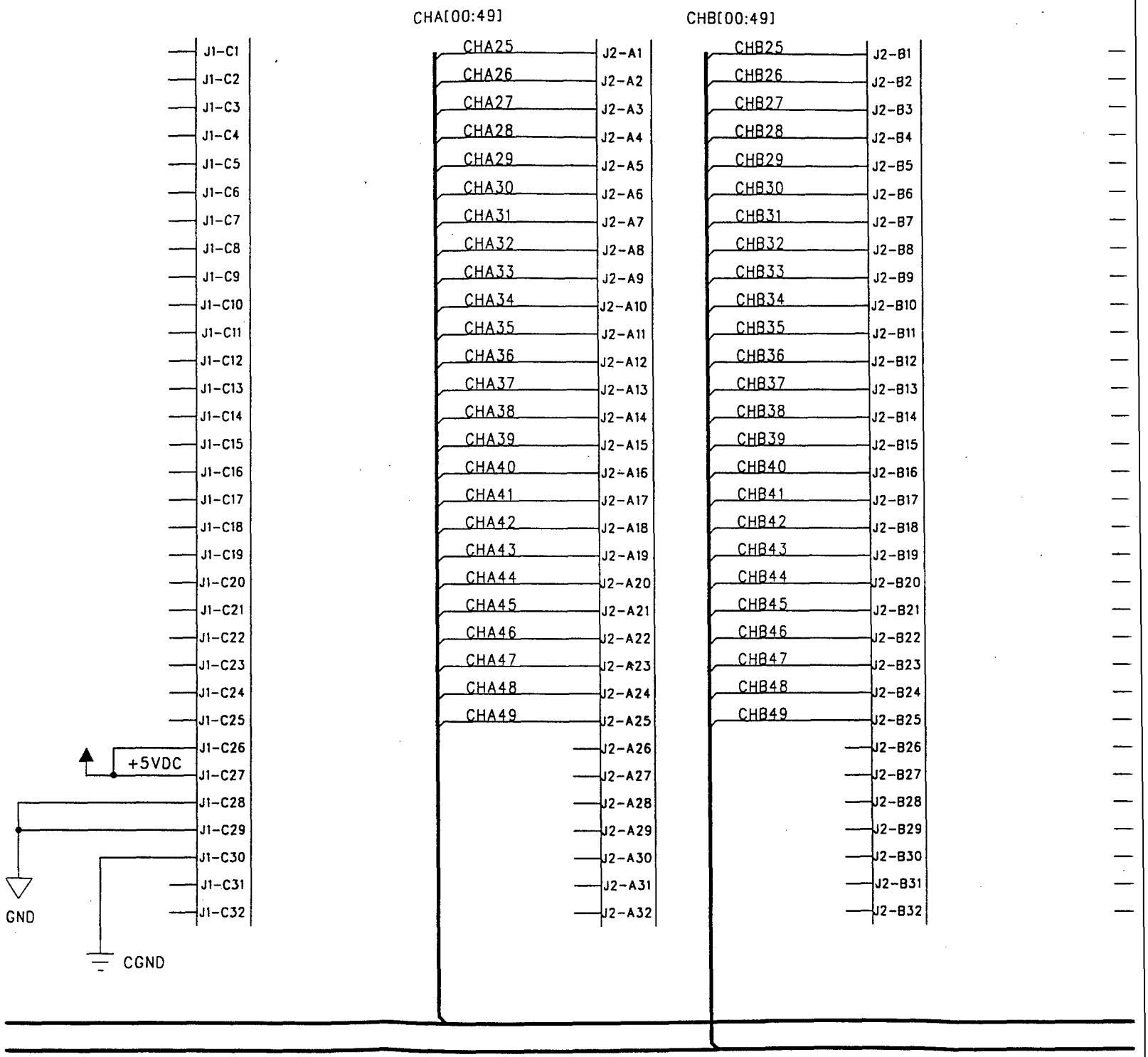
AS BUILT ON PCB #EDL9742

ELECTRONIC DESIGN LAB REDDING, CT 06896 (203) 938-2142	CLIENT	DIMENSIONAL MEDIA ASSOC	
	PROJECT	MOE DRIVER	REV A
	LOGIC BOARD CHANNELS 0 - 49 OUTPUT CONNECTORS		
	DRAWING BY	JEP	DATE 7-1-97
	DESIGN BY	JOG	
	JOB 601	DRAWING#	#: 5 OF 5



LOWER BACKPLANE CONNECTOR

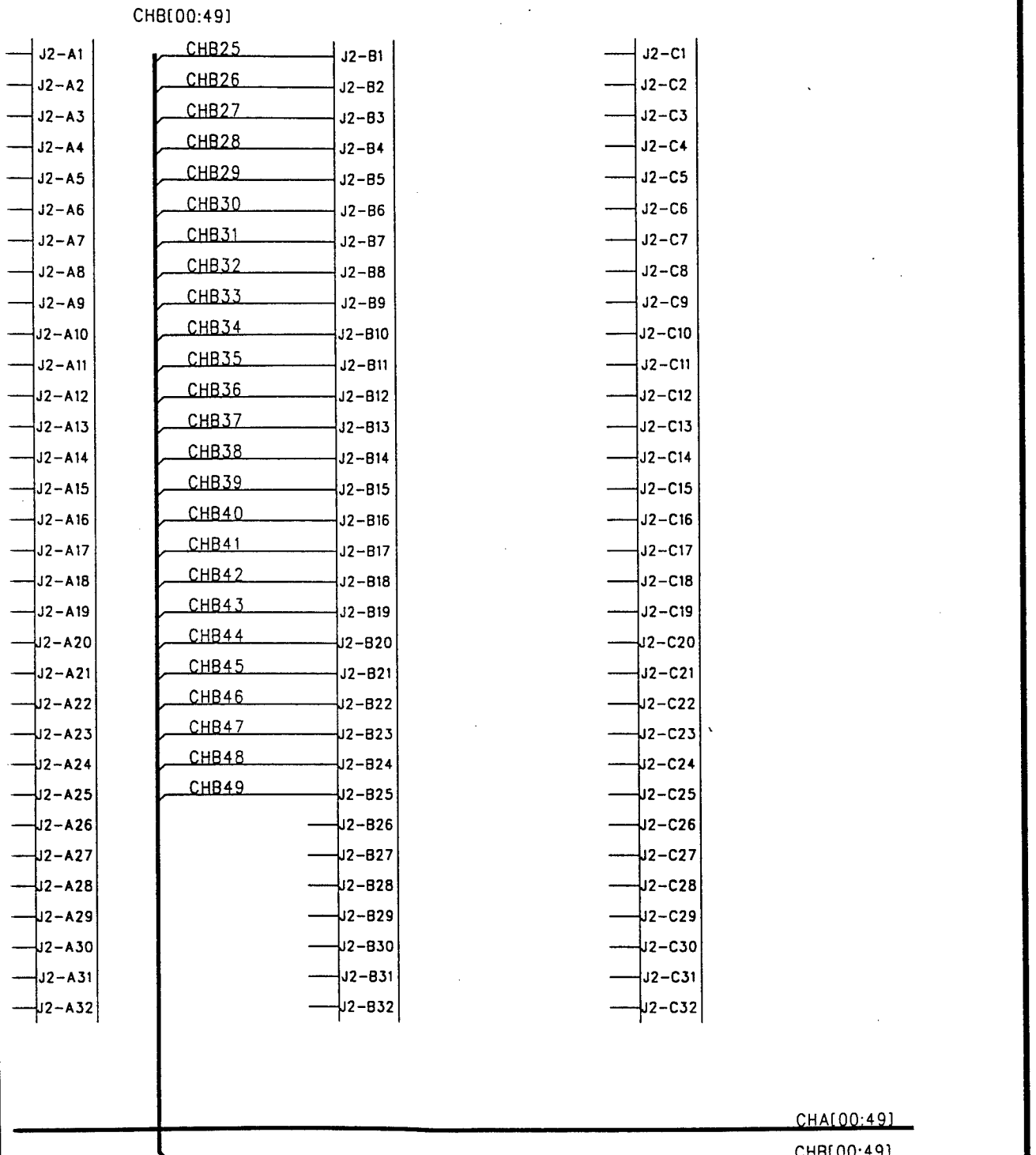
B



CONNECTOR

UPPER BACKPLANE CONNECTOR

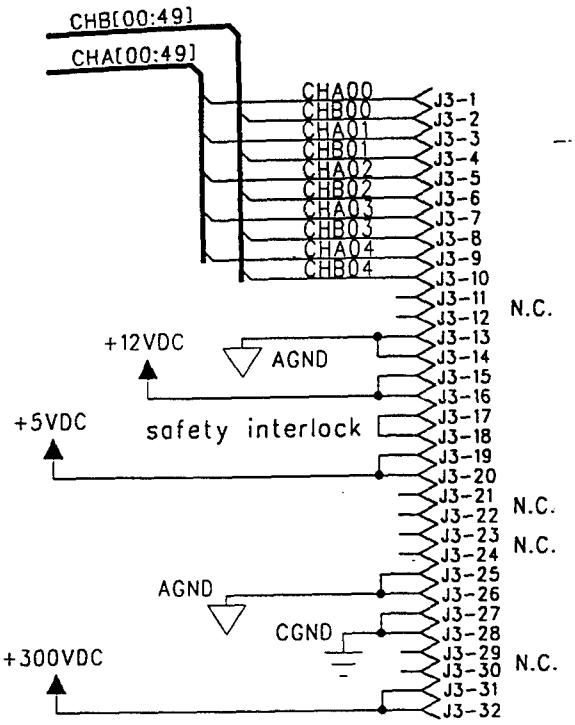
ELECTRONIC
DESIGN LAB
REDDING, CT
06896
(203)
938-2142

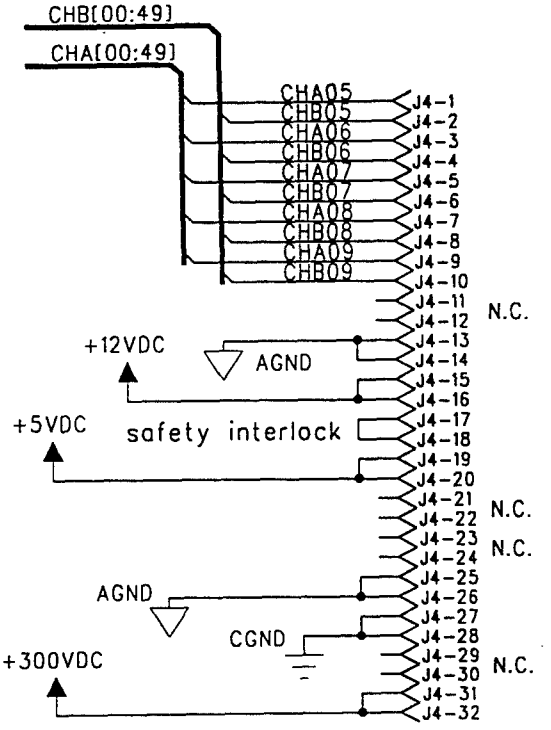


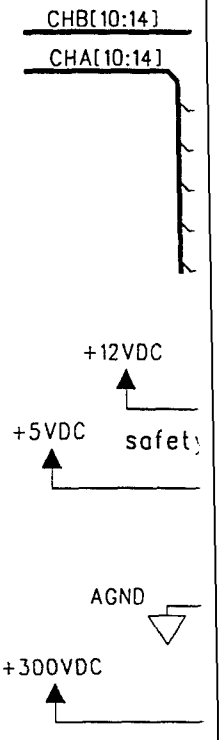
UPPER BACKPLANE CONNECTOR

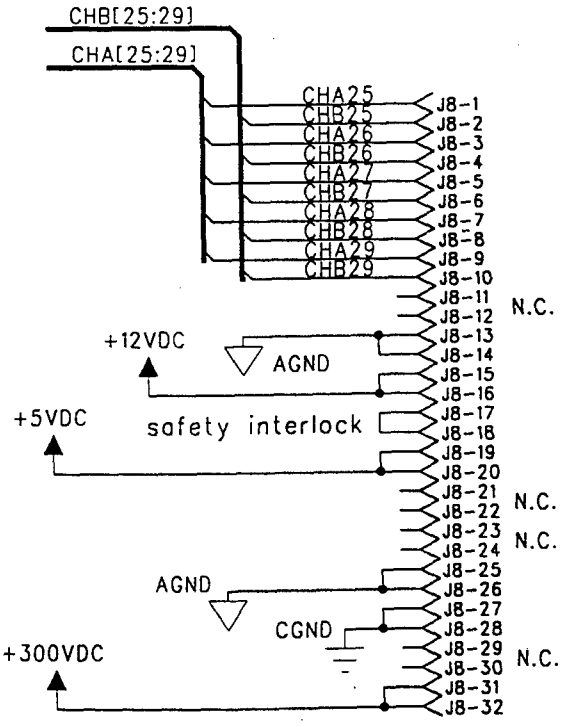
ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203)
938-2142

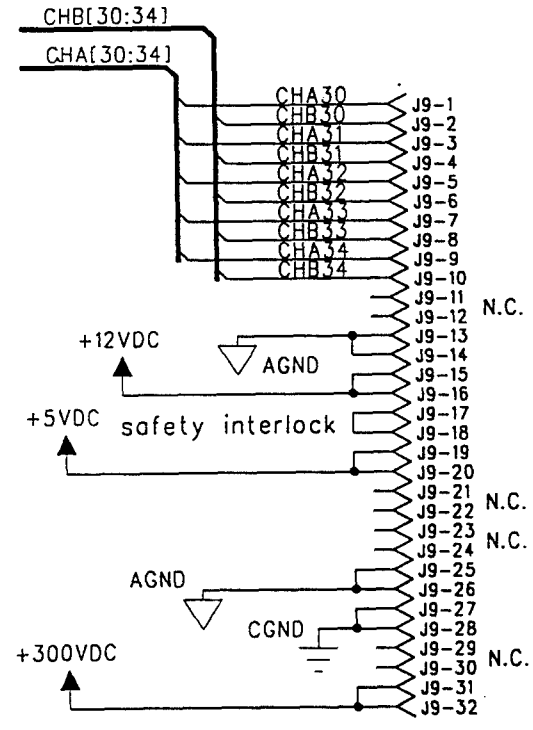
CLIENT	DIMENSIONAL MEDIA ASSOC		
PROJECT	MOE DRIVER		REV
MOE ELECTRONICS BACKPLANE			
DRAWING BY	MJA	DATE	10-20-97
DESIGN BY	MJA		
JOB	601	DRAWING#	#: 1 OF 2

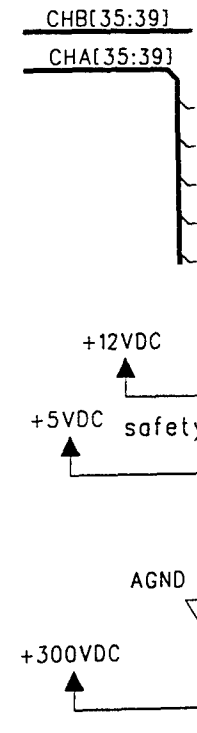




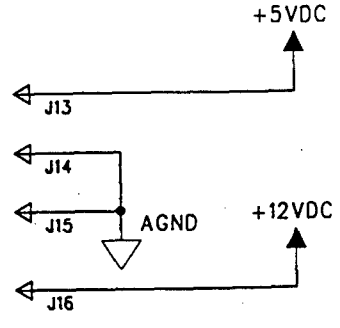


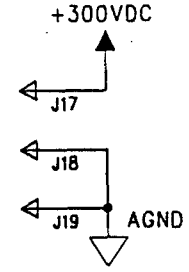






J13, 14, 16 connect to the linear power supply. J15 is a test point.



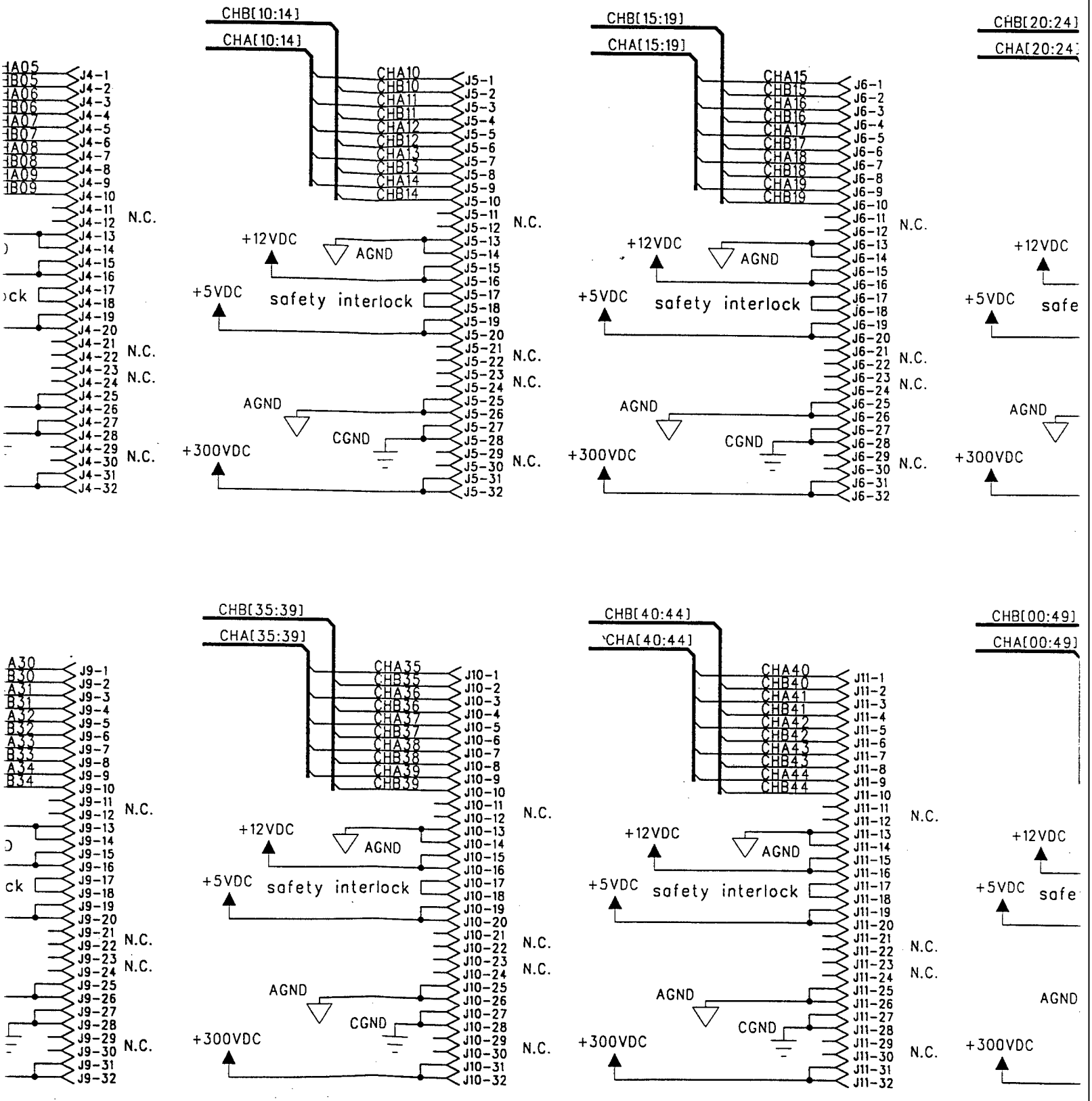


J17, 18 connect to the high voltage power supply.

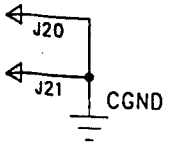
J19 connects to J15.

B

(1)

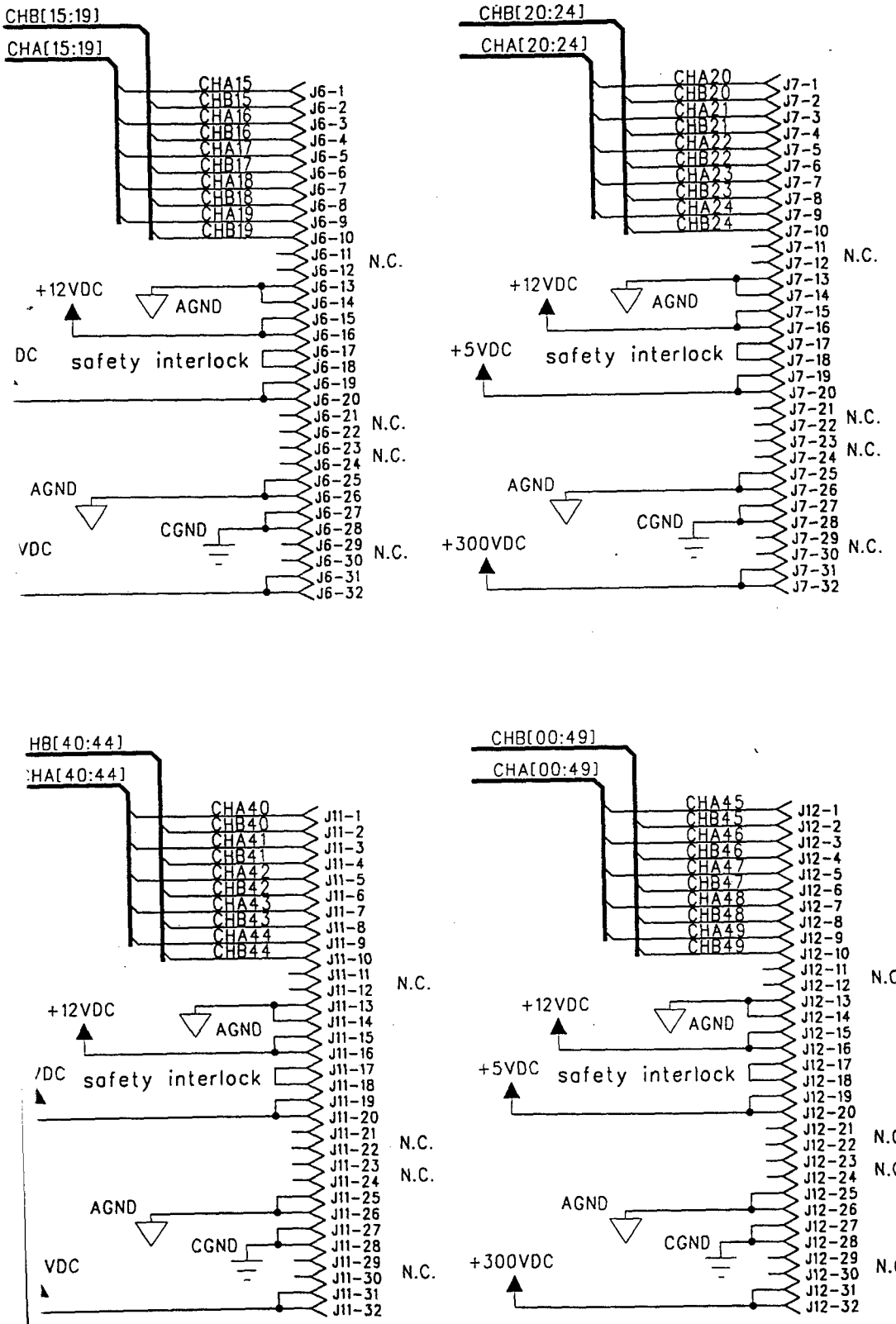


J17, 18 connect to the high voltage power supply.



TP = Test Point
CGND = Chassis Ground
AGND = Analog Ground

ELECTRO DESIGN L
REDDING,
06896
(203)
938-214



TP = Test Point
CGND = Chassis Ground
AGND = Analog Ground

ELECTRONIC DESIGN LAB
REDDING, CT
06896
(203) 938-2142

CLIENT	DIMENSIONAL MEDIA ASSOC		
PROJECT	MOE DRIVER		REV
MOE ELECTRONICS BACKPLANE			
DRAWING BY	MJA	DATE	10-20-97
DESIGN BY	MJA		
JOB 601	DRAWING#	#: 2 OF 2	